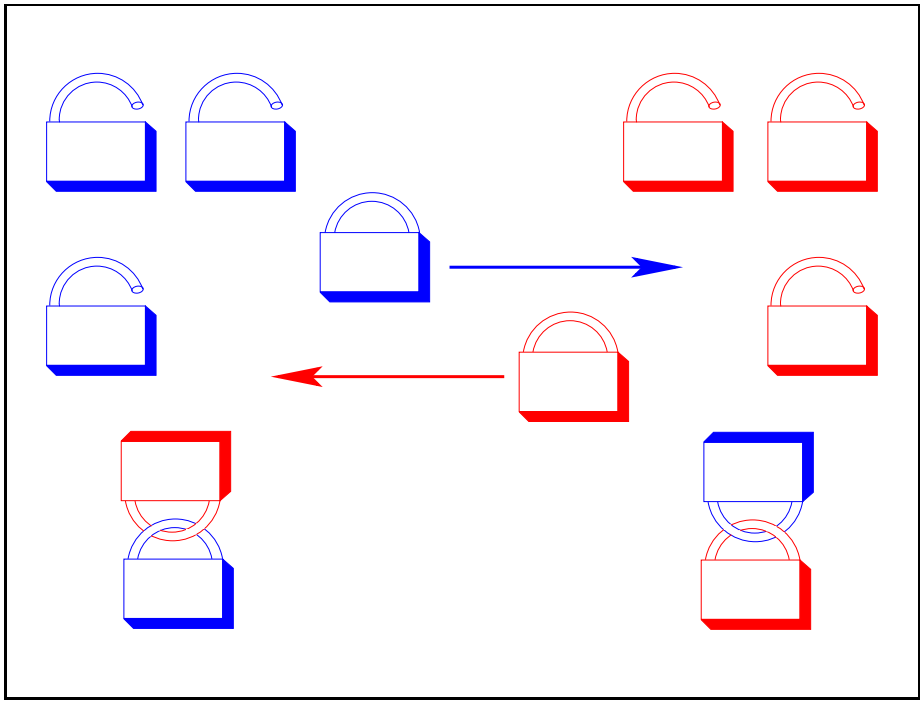
Data structures for parallel exponentiation in finite fields

Michael Nöcker
Universität Paderborn
Fachbereich 17 Mathematik-Informatik
33095 Paderborn



The drawing on the title page is an illustration of a mechanical analog of the Diffie-Hellman key exchange. This drawing is due to an idea of Maurer & Wolf.

Contents

1. Introduction

This thesis deals with *exponentiation* in finite fields $\mathbb{F}_{q^n}$. In particular, it discusses how exponentiation can benefit from *parallel computing*. Evaluation of a power of an element in $\mathbb{F}_{q^n}$ is of general interest. Currently most important, it is the basic arithmetic operation behind some *cryptographic primitives*. The security of some primitives is based e.g. on the *discrete logarithm problem*, see Section 2.2. The fields used in cryptography have to be large enough, so that arithmetic performance improvements are called for. Cryptography should not affect significantly the performance of the original application. Therefore it has to be fast, and it is desirable to speed up the basic arithmetic behind cryptography. This includes the efficient calculation of powers of elements in $\mathbb{F}_{q^n}$.

The progress on exponentiation throughout the last decade has brought up special properties of the field representation into the foreground. Computer scientists and engineers may consider the field representation as a choice of a particular data structure for the elements of a finite field. For mathematicians, it is a place to bring the rich properties of different representations of a finite field into play.

The two most common choices for a basis representation of $\mathbb{F}_{q^n}$ are *polynomial bases* and *normal bases*. Both have distinctive advantages and disadvantages in view of efficient arithmetics. The research of the last decade led to the fact that the first are currently associated with fast sequential arithmetic, whereas the latter are good for parallelizing exponentiation.

The question of attempting a synthesis of both representations lays at hand. A natural starting point is a characterization of the different properties of all possible bases with respect to parallel exponentiation. An exponentiation can be regarded as a sequence of multiplications and evaluations of the *Frobenius automorphism*. Polynomial bases offer a data structure that supports fast multiplication in $\mathbb{F}_{q^n}$. Normal bases yield zero cost for computing the Frobenius automorphism.

The work in hand considers these questions deeply. Its main conclusions are:

1. The ratio c between the cost for evaluating the Frobenius automorphism and the cost for a multiplication determines the benefit which can be taken from parallelizing exponentiation in a given basis representation. We prove lower and upper bounds on parallel exponentiation with respect to this ratio.

2. Generalized Gauß periods were suggested in a discussion that started with optimal normal bases. The latter one are prime Gauß periods of type 1 or 2. Gao *et al.* (2000) showed that prime Gauß periods can unite fast polynomial multiplication and zero-cost Frobenius. These prime Gauß periods were conceptually generalized by Feisel *et al.* (1999), motivated by the expectation to find larger classes of finite fields with good arithmetic

properties. The present work proves that general Gauß periods indeed enable fast multiplication and a zero-cost Frobenius for the expected classes of finite fields.

Of course, this thesis is not the first work which evaluates arithmetic in finite fields on the background of cryptography. The thesis of Wassermann (1993) presented arithmetic for different basis representations *in sequential*. Geiselmann (1994) developed a *hardware implementation* for exponentiation in finite fields. The aim of this thesis is to bring *parallel computing* and *general Gauß periods* into play.

As a first step we present a parallel algorithm for exponentiation. Exponentiation is related to the model of addition chains, see Knuth (1998). We use the more detailed concept of weighted addition chains with scalar, which have been described in von zur Gathen & Nöcker (2000) (Section 4.1). For this model, we also prove a *lower bound* for the cost of parallel exponentiation (Result 4.17), see Nöcker (2000). Both the algorithm and the lower bound fill the gap between two results of Kung (1976) and von zur Gathen (1991), which are special cases. This is the first corner stone of our work. It points the way to suitable data structures for efficient parallel exponentiation. The ratio c mentioned above serves as the criterion to decide which data structure to use. We determine this ratio for polynomial bases with selected moduli such as general and sparse irreducible polynomials in Section 5.2.

Arbitrary normal bases are discussed in Section 6. They show a high potential speed-up but poor performance. For normal bases, we reduce the *computation of the inverse* in $\mathbb{F}_{q^n}^\times$ to the calculation of a short(est) addition chain for $n-1$, see also von zur Gathen & Nöcker (1999). Section 6.5 generalizes ideas of Wang *et al.* (1985), Itoh & Tsujii (1988b), and others that are related to this topic. Our new Algorithm 6.27 computes the inverse in $\mathbb{F}_{q^n}$ in parallel. For a normal basis representation in $\mathbb{F}_{2^n}$ this algorithm uses only 2 processors. It also has optimal depth.

As the second corner stone of our work we utilize both fast polynomial multiplication and normal bases for *general Gauß periods*. As mentioned before, this type of normal elements has been introduced by Feisel *et al.* (1999). We shortly review the basics in Section 7. We achieve our new results in two steps. First in Section 8, we generalize an idea of Gao *et al.* (2000) to use polynomial multiplication for *prime power Gauß periods*, see Result 8.1. The underlying Algorithm 8.24 enables us to extend an algorithm of Wassermann (1993) on the computation of the *multiplication matrix*. We derive this matrix for prime power Gauß periods by following his idea.

In our second step in Section 9, we introduce *decomposable Gauß periods* as a helpful tool to emphasize the connection between the factorization of Gauß periods and the decomposition of the underlying finite field. A similar idea, but with different aims, may be found in Gao (2001). For such normal elements,

multiplication can be easily reduced to the case of prime power Gauß periods, see Result 9.1. With the help of this tool, we deduce an efficiently computable criterion whether normal Gauß periods exists for a presented $\mathbb{F}_{q^n}$.

Determining the parallel potential of a given data structure and revealing normal general Gauß periods as a suitable basis representation are the two main results of this thesis. As a final step we develop a scalable parallel version of Brauer's exponentiation algorithm in Section 10. It takes into account the restrictions of existing parallel computers such as a limited number of processors and communication delay. Finally, our implementations confirms the validity of our theoretically obtained results.

I wish to thank all people that have supported me during the preparation of this thesis. For inspiring discussions, I thank all collegues of the research group Algorithmische Mathematik and the participants of the Oberseminar Algorithmische Mathematik. My special thanks to Uwe Nagel for his comments on my view of Gauß periods. Many thanks to Thilo Pruschke for his careful reading. Last but not least, I want to thank my supervisor Prof. Joachim von zur Gathen for introducing me to the fascinating field of computer algebra, for his advice, and for the years in his research group at the Universität Paderborn.

2. Background

We compile some information which are the background of this work. Beside some basic facts about algebraic structures that we use frequently (Section 2.1), we describe some cryptographic protocols that heavily profit from fast exponentiation (Section 2.2). We include in Section 2.3 some information on the hardware and software we used for the experiments. These experiments are documented throughout the text. A reader familiar with these topics may skip this section.

2.1. Algebraic structures. We sketch the definitions and basic facts of the algebraic structures that are used in this work. For details and proofs we refer to the books of Jacobson (1974) and Lidl & Niederreiter (1983).

DEFINITION 2.1. *(i) A* monoid *or* semi-group *is a non-empty set $\mathcal{G}$ with a binary operation $\cdot\colon \mathcal{G} \times \mathcal{G} \to \mathcal{G}$ satisfying:*

- *Associativity:* $\forall A, B, C \in \mathcal{G}\colon (A \cdot B) \cdot C = A \cdot (B \cdot C)$,
- *Identity:* $\exists E \in \mathcal{G}\, \forall A \in \mathcal{G}\colon A \cdot E = E \cdot A = A$.

(ii) A group *is a monoid satisfying:*

- *Inverse:* $\forall A \in \mathcal{G} \exists A^{-1} \in \mathcal{G}\colon A \cdot A^{-1} = A^{-1} \cdot A = E$.

A group $\mathcal{G}$ is called Abelian *or* commutative *if*

- *Commutativity:* $\forall A, B \in \mathcal{G}\colon A \cdot B = B \cdot A$.

Our basic group is the commutative group $(\mathbb{Z}_r^\times, \cdot)$ of (multiplicative) units modulo $r \in \mathbb{N}_{\geq 2}$. A non-empty subset $\mathcal{U} \subseteq \mathcal{G}$ is a *subgroup* of $(\mathcal{G}, \cdot)$ if $\mathcal{U}$ together with the operation $\cdot$ is also a group. A group is *finite* if $\#\mathcal{G} \in \mathbb{N}_{\geq 1}$. The basic relation between the orders of $\mathcal{U}$ and $\mathcal{G}$ was given by Lagrange; see Jacobson (1974), Theorem 1.5, for a proof.

LAGRANGE'S THEOREM 2.2. *The order $\#\mathcal{U}$ of a subgroup $\mathcal{U}$ of a finite group $\mathcal{G}$ is a divisor of the order of $\mathcal{G}$.*

As a direct consequence we have the following result which is often dedicated to Fermat. In fact, he has discussed the case $(\mathbb{Z}_p^\times, \cdot)$ where $p \in \mathbb{N}_{\geq 2}$ is a prime.

FERMAT'S LITTLE THEOREM 2.3. *Let $\mathcal{G}$ be a finite group. Then $A^{\#\mathcal{G}} = 1$ for all $A \in \mathcal{G}$.*

For an element $A \in \mathcal{G}$ and a subgroup $\mathcal{U}$ of $\mathcal{G}$ we define $A\mathcal{U} = \{A \cdot U\colon U \in \mathcal{U}\}$ to be the *(right) coset* of $\mathcal{U}$. A subset $\{A_1, \ldots, A_m\}$ of a finite group $\mathcal{G}$ *generates* the subgroup $\{A_{j_1} \cdots A_{j_t}\colon 1 \leq j_1, \ldots, j_t \leq n\} = \langle A_1, \ldots, A_n\rangle$ of $\mathcal{G}$. A finite group $\mathcal{G}$ is *cyclic* if there exists an element $G \in \mathcal{G}$—which is called *primitive*—such that $\langle G\rangle = \mathcal{G}$.

DEFINITION 2.4. *A* ring *is a set* $\mathcal{R} \neq \emptyset$ *with two binary operations* $+\colon \mathcal{R} \times \mathcal{R} \to \mathcal{R}$ *and* $\cdot\colon \mathcal{R} \times \mathcal{R} \to \mathcal{R}$ *satisfying:*

- $(\mathcal{R}, +)$ *is a commutative group with identity* 0,
- $(\mathcal{R}, \cdot)$ *is a monoid with identity* 1,
- *Distributive law:* $\forall A, B, C \in \mathcal{R}\colon A \cdot (B+C) = A \cdot B + A \cdot C$ *and* $(A+B) \cdot C = A \cdot C + B \cdot C$.

Our basic example for a ring is $(\mathbb{Z}_r^{\times}, +, \cdot)$. A *ring homomorphism* from a ring $\mathcal{R}$ to a ring $\mathcal{R}'$ is a map φ such that $\varphi(R_1 + R_2) = \varphi(R_1) + \varphi(R_2)$, $\varphi(R_1 \cdot R_2) = \varphi(R_1) \cdot \varphi(R_2)$ for all $R_1, R_2 \in \mathcal{R}$, and furthermore $\varphi(1) = 1$. If φ is surjective then it is called an *epimorphism*. A bijective φ is an *isomorphism*. If $\mathcal{R} = \mathcal{R}'$ then the isomorphism is also called *automorphism*.

DEFINITION 2.5. *A* finite field $\mathbb{F}$ *is a finite ring satisfying:*

- $(\mathbb{F} \setminus \{0\}, \cdot)$ *is a commutative group.*

FACT 2.6 (Lidl & Niederreiter 1983, Theorem 2.2 and Theorem 2.5). *The order of any finite field is a prime power* $q \in \mathbb{N}_{\geq 2}$. *Given any prime power* q, *there exists up to isomorphism exactly one finite field of order* q.

We denote this finite field of order q by $\mathbb{F}_q$. If $q = p^e$ for a prime p then the field $\mathbb{F}_q$ has *characteristic* $\mathrm{char}(\mathbb{F}_q) = p$. The field $\mathbb{F}_p$ is the *prime (sub)field* of $\mathbb{F}_q$.

FACT 2.7 (Lidl & Niederreiter 1983, Theorem 2.8). *In the field* $\mathbb{F}_q$, *the multiplicative group* $\mathbb{F}_q^{\times} = \mathbb{F}_q \setminus \{0\}$ *is cyclic.*

Let q^n be a prime power for $n \in \mathbb{N}_{\geq 1}$. Then $\mathbb{F}_q$ is a *subfield* of $\mathbb{F}_{q^n}$. We may regard $\mathbb{F}_{q^n}$ as an $\mathbb{F}_q$*-vector space* of dimension $[\mathbb{F}_{q^n} : \mathbb{F}_q] = n$ with basis $\mathcal{B} = (\alpha_0, \ldots, \alpha_{n-1})$. Thus any element of $\mathbb{F}_q$ can be written as a uniquely determined *linear combination* of the elements of $\mathcal{B}$ over $\mathbb{F}_q$, i.e. $\mathbb{F}_{q^n} = \{\sum_{0 \leq i < n} A_i \alpha_i \colon A_i \in \mathbb{F}_q\}$.

The map

$$\sigma\colon \mathbb{F}_{q^n} \to \mathbb{F}_{q^n} \text{ with } \sigma(A) = A^q$$

is an isomorphism. It is called the *Frobenius automorphism* and satisfies (see Lidl & Niederreiter (1983), p. 53)

$$\begin{array}{lcll} \sigma(A+B) & = & \sigma(A) + \sigma(B) & \text{and,} \\ \sigma(A \cdot B) & = & \sigma(A) \cdot \sigma(B) & \text{for all } A, B \in \mathbb{F}_{q^n}, \text{ and} \\ \sigma(A) & = & A & \text{if and only if } A \in \mathbb{F}_q. \end{array} \tag{2.8}$$

The *Galois group* $\mathrm{Gal}(\mathbb{F}_{q^n}/\mathbb{F}_q)$ is the set of all field automorphisms σ' of $\mathbb{F}_{q^n}$ that fix $\mathbb{F}_q$ element-wise.

FACT 2.9 (Bosch 1993, Section 3.9). *The Galois group* $\mathrm{Gal}(\mathbb{F}_{q^n}/\mathbb{F}_q)$ *has order* n *and is cyclic. It is generated by the Frobenius automorphism* $\sigma\colon \mathbb{F}_{q^n} \to \mathbb{F}_{q^n}$ *with* $\sigma(A) = A^q$.

See also Lidl & Niederreiter (1983), Theorem 2.21, and Jacobson (1974), Section 4.13, for proofs.

2.2. Cryptographic background. We have already mentioned the connection between exponentiation and cryptography. We shortly summarize some protocols over finite fields that make use of exponentiation. We refer to the book of Stinson (1995) for detailed presentations.

Public key cryptography. Cryptography, i.e. the art of hiding information to eavesdroppers and other people but the dedicated receiver, has a long history from its ancient roots up to our time. Classical cryptography uses a prearranged secret piece of information K called the *key*, which is not only used to encrypt a message but which also allows the receiver to decrypt the encoded message easily. This is also called a *symmetric cryptosystem*. Modern telecommunication, in particular the fast-growing Internet and electronic commerce, call for security that is based on practicable cryptography which dispense with such a prearranged secret K. Diffie & Hellman (1976) presented the idea of *public key cryptography* to meet this demand. In a public key cryptosystem encryption and decryption are governed by distinct keys, a *public key* K_e for encryption and a *private key* K_d for decryption. Computing the private key K_d from the public one K_e has to be *computationally infeasible*[1].

Diffie-Hellman key exchange. The core arithmetic operation for the *key exchange protocol* dedicated to Diffie & Hellman (1976) is exponentiation in a finite field $\mathbb{F}_q$. Originally, the authors have restricted their description to prime fields with q a large prime, but it can easily be adapted to all finite fields $\mathbb{F}_{q^n}$ with $n \in \mathbb{N}_{\geq 1}$, or to other multiplicative groups $\mathcal{G}$ for which the *discrete logarithm problem* is difficult to solve; see e.g. Menezes *et al.* (1993), Section 6, for a discussion of the discrete logarithm problem.

DEFINITION 2.10. *Let* $\mathcal{G}$ *be a group and* A, G *be elements of* $\mathcal{G}$ *such that* $A \in \langle G \rangle$. *The task to find an integer* $x \in \mathbb{Z}$ *such that* $G^x = A$ *is called the* discrete logarithm problem.

[1] Diffie & Hellman "call a task *computationally infeasible* if its cost as measured by either the amount of memory used or the runtime is finite but impossible large.", Diffie & Hellman (1976), p. 646.

PROTOCOL 2.11. Diffie-Hellman key exchange.

Public: A prime power $q^n \in \mathbb{N}_{\geq 2}$ and a primitive element $G \in \mathbb{F}_{q^n}^\times$.
Goal: Agreement upon a common secret key $K \in \mathbb{F}_{q^n}^\times$ using an insecure channel[2].

1. Alice chooses an integer $k_A \in \{1, \ldots, q^n - 1\}$ at random. She computes $A = G^{k_A} \in \mathbb{F}_{q^n}$ *by exponentiation.*
2. Bob chooses an integer $k_B \in \{1, \ldots, q^n - 1\}$ at random. He computes $B = G^{k_B} \in \mathbb{F}_{q^n}$ *by exponentiation.*
3. Alice and Bob exchange A and B via the insecure channel.
4. Alice computes $K = B^{k_A} \in \mathbb{F}_{q^n}$ *by exponentiation.*
5. Bob computes $K = A^{k_B} \in \mathbb{F}_{q^n}$ *by exponentiation.*

Obviously, the security of this protocol is based on the fact that for an eavesdropper Eve it is computationally infeasible to get K from the transmitted elements $A, B \in \mathbb{F}_{q^n}$ and the public element $G \in \mathbb{F}_{q^n}$. If Eve could solve the discrete logarithm problem in $\mathbb{F}_{q^n}$ then she can get k_A from A and thus $K = B^{k_A}$, performing Alice's last step of the protocol. It is still an open question whether breaking the Diffie-Hellman key exchange is as hard as solving the discrete logarithm problem, but see Maurer & Wolf (1999).

ElGamal's public key system. Diffie & Hellman (1976) proposed their system for key distribution only. The generated key can then be used to establish an arbitrary symmetric cryptosystem. ElGamal (1985) developed a public key cryptosystem that is based on the idea of Protocol 2.11.

PROTOCOL 2.12. ElGamal public key system.

Public: A prime power $q^n \in \mathbb{N}_{\geq 2}$ and a primitive element $G \in \mathbb{F}_{q^n}^\times$.
Goal: Secure transmission of a message $M \in \mathbb{F}_{q^n}$ between two parties.

1. Bob chooses an integer $k_B \in \{1, \ldots, q^n - 1\}$ at random. He computes $K_B = G^{k_B} \in \mathbb{F}_{q^n}$ *by exponentiation* and publishes K_B as his public key.
2. Alice chooses an integer $k \in \{1, \ldots, q^n - 1\}$ at random. She computes $K = K_B^k \in \mathbb{F}_{q^n}$ and $K_A = G^k \in \mathbb{F}_{q^n}$ *by exponentiation.*
3. Alice encrypts the message $M \in \mathbb{F}_{q^n}^\times$ by computing $C = M \cdot K$. She sends (C, K_A) to Bob.
4. Bob first computes $K = K_A^{k_B} \in \mathbb{F}_{q^n}$ *by exponentiation.* He subsequently performs the division C/K to get M.

Again, solving the discrete logarithm problem would break this cryptosystem.

[2] An *insecure channel* marks that all information exchanged between Alice and Bob are assumed to be public. We do not deal with the more complicated property that the exchanged data may also be manipulated by a third person during the transmission.

Authentication. The (assumed) difficulty of solving the discrete logarithm problem in finite fields can also be applied to authenticate a login in a multiuser system without really storing the passwords, see the remark in McCurley (1990). Assume that a primitive element $G \in \mathbb{F}_{q^n}$ is given in advance. Without loss of generality, we assume that the password m is an integer in $\{1, \ldots, q^n - 1\}$. Then the system computes $M = G^m \in \mathbb{F}_{q^n}$ *by exponentiation* and stores M instead of m. Recovering the password from M is just solving the discrete logarithm problem. Now every time a user logs in with password $m' \in \{1, \ldots, q^n - 1\}$ the system computes $G^{m'} \in \mathbb{F}_{q^n}$ *by exponentiation* and compares the result with M. There is no need to store the password itself.

All the described cryptographic applications are additional computation if we share a user's point of view. Thus, it is a crucial task to speed up these applications. This can be done by speeding up exponentiation in the finite field $\mathbb{F}_{q^n}$.

Security of the discrete logarithm. The intention of this work is to discuss some aspects of those arithmetic which is used in cryptographic applications. Therefore, we do not discuss security in the following chapters but we are interested in algorithms. All examples and test series are chosen with the intention to illustrate arithmetic properties. Nevertheless, we shortly cite some facts to give a feeling what *security* may mean with respect to the discrete logarithm problem.

Starting with the paper of Odlyzko (1985), there are several surveys on the discrete logarithm problem, e.g. McCurley (1990), van Oorshot (1992), and Odlyzko (1994, 2000). The discrete logarithm problem for $\mathbb{F}_{2^n}$ can be solved in subexponential time with

$$\exp\left((c + o(1))n^{1/3}\log^{2/3} n\right)$$

where $c \in \mathbb{R}_{\geq 0}$ is a positive constant. The corresponding algorithm follows the approach of the so-called *index calculus method*[3]. Reports on implementations propagate that cryptography based on the discrete logarithm problem is now insecure for $\mathbb{F}_{2^n}$ with $n = 127$ (see Blake *et al.* (1984), Coppersmith & Davenport (1985)), and with $n \in \{227, 313, 401\}$ (see Gordon & McCurley (1992)). Gordon & McCurley (1992) also report on experiments that show that discrete logarithms in $\mathbb{F}_{2^{503}}$ and $\mathbb{F}_{2^{521}}$ are feasible[4].

An algorithmic approach of Pohlig & Hellman (1978) restricts the recommended finite fields $\mathbb{F}_{q^n}$ to those whose order $\#\mathbb{F}_{q^n}^{\times} = q^n - 1$ has at least one large prime factor. We summarize our short survey on the cryptographic background using Odlyzko's words: "The lesson to be drawn from these algorithms is that

[3] For an introduction on the index calculus method see e.g. Menezes *et al.* (1993), Section 6.6.

[4] "We believe that 521 should now be possible to complete, albeit with the consumption of massive amounts of computing time. Discrete logarithms in $GF(2^{593})$ still seem to be out of reach.", Gordon & McCurley (1992), p. 322.

to have a secure public key cryptosystem, one needs to choose the field $[\mathbb{F}_{q^n}]$ carefully."[5]

2.3. Environment of the experiments. One goal of this work is to discuss parallel exponentiation from a practical point of view. Thus, we implemented our main algorithmic ideas in software and will present times and experimental comparisons throughout the text.

All experiments were performed on the *Siemens hpcLine*[6] computer, or *PSC2* for short, at the *Paderborn Center for Parallel Computing* (PC^2). This machine is a cluster system with 96 nodes and 192 processors. Each node contains 2 Pentium II processors rated at 450 MHz, and has 512 MByte of local memory. The operating system is RedHat 6.2 Linux. The nodes are connected to the network via Dolphin PCI/SCI interfaces. We use a communication called ScaMPI which has a rate of 83 MBytes/s and 8 μs latency. The network itself consists of a mesh of 12 times 6 nodes.

We have used the software library PUB to include parallel computation in our `C++`-code. PUB supports parallel computation on computer networks and massive-parallel machines. It supports implementations that follow the ideas of the BSP-model invented by Valiant (1990), see Section 10.1. We have used version 7 of the PUB library which has been developed by the group of Professor Meyer auf der Heide at the University of Paderborn.[7]

[5] Odlyzko (1994), p. 270.

[6] The cited facts and more details on this machine are available via `http://www.upb.de/pc2/services/systems/psc/main.html`.

[7] Source code and manuals are available via `http://www.uni-paderborn.de/~pub/`. Many special thanks to Ingo Rieping and Olaf Bonorden for their support on PUB.

3. Sequential exponentiation and addition chains

We start with recalling some classical sequential algorithms for exponentiation. They serve as an initial step for parallelization. In the first part (Section 3.1), we relate exponentiation to addition chains. Addition chains can be regarded as a model of computation: the element 1 is initially given, and new elements can be computed by adding two already calculated predecessors. The additions are related to multiplication in the original exponentiation algorithm. Some observations have inspired us to extend the model by a second operation: Doubling a point on an elliptic curve is different from adding two distinct points. Computing the Frobenius automorphism in a finite field can often be done much faster than by successive multiplication. To model this, we add multiplication of an element by a scalar $q \in \mathbb{N}_{\geq 2}$ as a new operation for addition chains. These q-addition chains will become our basic model throughout this work. We adjoin different weights to both types of steps to be in touch with the underlying multiplicative group. The measure of the efficiency of an algorithm for powering is the length of the corresponding weighted addition chain with scalar.

We reformulate and analyze Brauer's idea to produce good, i.e. short, addition chains in Section 3.2. Next, we consider an algorithm of Brickell *et al.* (1993); Yao (1976) described a similar algorithm for the computation of addition chains for sets. We describe both versions of this algorithm in Section 3.3 and give upper bounds on the length of the corresponding q-addition chains. In the final part of this section (Section 3.4), we compare all discussed addition chain algorithms. These experiments show that different types of steps gain advantage for sequential exponentiation. In particular, Brauer's algorithm beats the competitors surprisingly clear in our model for certain weights.

3.1. Exponentiation and q-addition chains.

The exponentiation problem. Let $\mathcal{G}$ be a monoid. One may think of $\mathcal{G}$ as the multiplicative monoid in a ring $\mathcal{R}$ or the multiplicative group $\mathbb{F}_{q^n}^{\times}$ of the finite field with q^n elements. Let A be an element of $\mathcal{G}$ and $e \in \mathbb{N}_{\geq 1}$. Raising the *basis* A to the *exponent* e is then the calculation of

$$A^e = \underbrace{A \cdots A}_{e \text{ times}} \in \mathcal{G}.$$

In what follows we often assume $\mathcal{G}$ to be a finite group. In this case we have $A^{\#\mathcal{G}} = 1$ in $\mathcal{G}$ by Fermat's Little Theorem 2.3, and we may restrict the exponent to be $0 \leq e < \#\mathcal{G}$. We define $A^0 = 1 \in \mathcal{G}$.

Since $\mathcal{G}$ is associative the exponents are additive, that is for $e, f \in \mathbb{N}_{\geq 1}$ we have

$$A^{e+f} = \underbrace{A \cdots A}_{e+f} = (\underbrace{A \cdots A}_{e}) \cdot (\underbrace{A \cdots A}_{f}) = A^e \cdot A^f$$

and we can abstract from the concrete basis A and even from the given set $\mathcal{G}$ by identifying a multiplication of the form $A^e \cdot A^f$ in $\mathcal{G}$ with the addition $e + f$ of the two exponents e and f. This idea leads to a sequence of (positive) integers. Such a sequence contains 1, which represents the element A. All other included elements are the sum of two (not necessarily distinct) integers that appear in the sequence. Therefore, we reduce the exponentiation problem to the theory of *addition chains*[8].

Original addition chains. The standard survey on original addition chains can be found in the book of Knuth (1998), Section 4.6.3. Other overviews are given e.g. by von zur Gathen & Nöcker (1997) and Gordon (1998). An addition chain γ is a sequence of integers $a_0, \ldots, a_L$ with $a_0 = 1$ and $a_L = e$, and for all $1 \leq i \leq L$ there exist $0 \leq j, k < i$ such that $a_i = a_j + a_k$. Such an addition chain is said to *compute* $e \in \mathbb{N}_{\geq 1}$ if $e \in \{a_0, \ldots, a_L\}$. This definition does not specify the concrete predecessors to compute an a_i.

EXAMPLE 3.1. The sequence $1, 2, 3, 5, 6, 11$ is an addition chain computing $e = 11$. We can generate this sequence by computing 1, $2 = 1+1$, $3 = 2+1$, $5 = 3+2$, $6 = 5 + 1$, $11 = 6 + 5$. Alternatively one may get 6 as $6 = 3 + 3$ which does not touch the sequence of integers $1, 2, 3, 5, 6, 11$. ◊

This example shows that an addition chain is not uniquely determined by the sequence of exponents. Thus, we prefer the following definition taken from von zur Gathen & Nöcker (1999) which does not restrict to the semantics but also includes the syntax.

DEFINITION 3.2 (Addition chains). *An addition chain γ is a sequence of pairs of non-negative integers $(j(1), k(1)), \ldots, (j(L), k(L))$ such that $0 \leq k(i) \leq j(i) < i$ for $1 \leq i \leq L$. We define the* semantics $\mathcal{S}(\gamma) = \{a_0, \ldots, a_L\}$ *by*

$$\begin{array}{lcll} a_0 & = & 1 & \text{and} \\ a_i & = & a_{j(i)} + a_{k(i)} & \text{for all } 1 \leq i \leq L. \end{array}$$

We say that γ computes $e \in \mathbb{N}_{\geq 1}$ if $e \in \mathcal{S}(\gamma)$.

We may resort the sequence and remove pairs $(j(i'), k(i'))$ for which exists an index $i \neq i'$ such that $a_i = a_{i'}$. By doing so we may always assume $1 = a_0 < a_1 < \ldots < a_L$. Then there is a bijection from γ onto $\mathcal{S}(\gamma) \times \mathcal{S}(\gamma)$ by $(j(i), k(i)) \mapsto (a_{j(i)}, a_{k(i)})$. We call $\mathcal{R}(\gamma) = \{(a_{j(1)}, a_{k(1)}), \ldots, (a_{j(L)}, a_{k(L)})\}$ the *set of rules* of γ. For better reading, we frequently describe γ in terms of $\mathcal{S}(\gamma)$ and $\mathcal{R}(\gamma)$.

[8] Addition chains (the original term is the German word *Additionsketten*) first appeared in Aufgabe 253 by Scholz (1937).

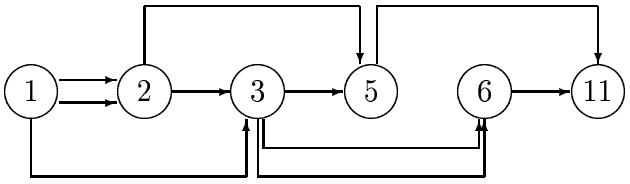
EXAMPLE 3.1 CONTINUED. The second way to compute the sequence $1, 2, 3, 5, 6, 11$ in the example above is described by the addition chain $\gamma = ((0,0), (1,0), (2,1), (2,2), (4,3))$: We have $a_0 = 1$ by definition, and get the semantics by successively inserting the given pairs as indices of the summands $a_{j(i)}$ and $a_{k(i)}$:

$$a_1 = a_0 + a_0 = 1 + 1 = 2, \qquad a_2 = a_1 + a_0 = 2 + 1 = 3,$$
$$a_3 = a_2 + a_1 = 3 + 2 = 5, \qquad a_4 = a_2 + a_2 = 3 + 3 = 6,$$
$$a_5 = a_4 + a_3 = 6 + 5 = 11.$$

The semantics is thus $\mathcal{S}(\gamma) = \{1, 2, 3, 5, 6, 11\}$ and the addition chain γ computes e.g. $e = 11$. The set of rules can be seen directly from each equation given above: $\mathcal{R}(\gamma) = \{(1,1), (2,1), (3,2), (3,3), (6,5)\}$. ◊

An element of $\mathcal{S}(\gamma)$ may also be called a *node* since a natural step is to illustrate addition chains by connected directed graphs[9]. The set of nodes is just $\mathcal{S}(\gamma)$. Each rule $(a_{j(i)}, a_{k(i)}) \in \mathcal{R}(\gamma)$ with $a_i = a_{j(i)} + a_{k(i)}$ represents two edges $(a_{j(i)}, a_i)$ and $(a_{k(i)}, a_i)$. We assume these edges to be different even if $a_{j(i)} = a_{k(i)}$. The case $j(i) = k(i)$ is also called a *doubling*.

EXAMPLE 3.1 CONTINUED. The following is a graph of the addition chain $\gamma = ((0,0), (1,0), (2,1), (2,2), (4,3))$ described above. The addition chain has length $L = 5$. In fact, there exists no shorter (original) addition chain for $e = 11$.



◊

The integer $L = \#\mathcal{S} - 1$ is called the *(sequential) length* $\ell_2(\gamma)$ of γ. Furthermore, we define the minimal length $\ell_2(e)$ for $e \in \mathbb{N}_{\geq 1}$ as

$$\ell_2(e) = \min\{\ell_2(\gamma) \colon \gamma \text{ computes } e\}. \tag{3.3}$$

If $k(i) = i - 1$ for $1 \leq i \leq L$ then γ is called a *star addition chain*. In this case the graph of γ has a path from a_0 to a_L of length L. We define

$$\ell_2^\star(e) = \min\{\ell_2(\gamma) \colon \gamma \text{ is a star addition chain which computes } e\}. \tag{3.4}$$

Let γ and δ be two addition chains. The *union* $\varepsilon = \gamma \cup \delta$ is given by $\mathcal{S}(\varepsilon) = \mathcal{S}(\gamma) \cup \mathcal{S}(\delta)$ and $\mathcal{R}(\varepsilon) = \mathcal{R}(\gamma) \cup \mathcal{R}(\delta)$. This gives $\ell_2(\varepsilon) \leq \ell_2(\gamma) + \ell_2(\delta)$, and $\mathcal{S}(\varepsilon)$ contains $\max\{a_{L(\gamma)}, b_{L(\delta)}\}$. We will also use the *concatenation* $\varepsilon = \gamma \odot \delta$ of two addition chains. This is given by $\mathcal{S}(\varepsilon) = \mathcal{S}(\gamma) \cup \{a_{L(\gamma)} \cdot b_i \colon b_i \in \mathcal{S}(\delta)\}$ and $\mathcal{R}(\varepsilon) = \mathcal{R}(\gamma) \cup \{(a_{L(\gamma)} \cdot b_{j(i)}, a_{L(\gamma)} \cdot b_{k(i)}) \colon (b_{j(i)}, b_{k(i)}) \in \mathcal{R}(\delta)\}$. In this case, we have $\ell_2(\varepsilon) = \ell_2(\gamma) + \ell_2(\delta)$ and $\mathcal{S}(\varepsilon)$ contains $a_{L(\gamma)} \cdot b_{L(\delta)}$.

[9] See e.g. Knuth (1998), p. 480–481.

EXAMPLE 3.5. Let γ be an addition chains given by $\mathcal{S}(\gamma) = \{1,2,4,5\}$ and $\mathcal{R}(\gamma) = \{(1,1),(2,2),(4,1)\}$. A second addition chain δ is described by the sets $\mathcal{S}(\delta) = \{1,2,3,6\}$ and $\mathcal{R}(\delta) = \{(1,1),(2,1),(3,3)\}$. The union addition chain $\varepsilon_1 = \gamma \cup \delta$ of γ and δ is defined by

$$\begin{aligned}\mathcal{S}(\varepsilon_1) &= \mathcal{S}(\gamma) \cup \mathcal{S}(\delta) = \{1,2,4,5\} \cup \{1,2,3,6\} = \{1,2,3,4,5,6\} \text{ and}\\ \mathcal{R}(\varepsilon_1) &= \mathcal{R}(\gamma) \cup \mathcal{R}(\delta) = \{(1,1),(2,2),(4,1)\} \cup \{(1,1),(2,1),(3,3)\}\\ &= \{(1,1),(2,1),(2,2),(4,1),(3,3)\}.\end{aligned}$$

The concatenation $\varepsilon_2 = \gamma \odot \delta$ extends the set of computed elements; the semantics of the resulting addition chain is

$$\mathcal{S}(\varepsilon_2) = \{1,2,4,5\} \cup \{5\cdot 1, 5\cdot 2, 5\cdot 3, 5\cdot 6\} = \{1,2,4,5,10,15,30\}.$$

The set of rules to generate the semantics $\mathcal{S}(\varepsilon_2)$ is given by

$$\begin{aligned}\mathcal{R}(\varepsilon_2) &= \{(1,1),(2,2),(4,1)\} \cup \{(5\cdot 1, 5\cdot 1),(5\cdot 2, 5\cdot 1),(5\cdot 3, 5\cdot 3)\}\\ &= \{(1,1),(2,2),(4,1),(5,5),(10,5),(15,15)\},\end{aligned}$$

and ε_2 is an addition chain for $e = 5\cdot 6 = 30$. ◊

Weighted addition chains with scalar. The special case of exponentiation in a finite field $\mathbb{F}_{q^n}$ motivated von zur Gathen (1991) to extend the former definition. Sometimes the *Frobenius automorphism* in $\mathbb{F}_{q^n}$ can be computed faster than by successive multiplications. To profit from this fact, we introduce a second type of operation for addition chains, see von zur Gathen & Nöcker (2000).

DEFINITION 3.6 (Weighted addition chains with scalar). *Let q be a positive integer, $q \geq 2$.*

(i) An addition chain with scalar q*, or q-addition chain for short, γ is a sequence of pairs of integers $(j(1),k(1)),\ldots,(j(l),k(l))$ such that $0 \leq k(i) \leq j(i) < i$, or $k(i) = -q$ and $0 \leq j(i) < i$ for $1 \leq i \leq l$. We define the semantics $\mathcal{S}(\gamma) = \{a_0,\ldots,a_l\}$ by*

$$\begin{array}{lcll} a_0 & = & 1 & \text{and}\\ a_i & = & a_{j(i)} + a_{k(i)} & \text{if } k(i) \neq -q, \text{ and}\\ a_i & = & q \cdot a_{j(i)} & \text{if } k(i) = -q.\end{array}$$

We call a pair $(j(i),-q)$ a q-step *and a pair $(j(i),k(i))$ with $k(i) \neq -q$ a* non-q-step.

(ii) A weight (c_Q, c_A) for γ is an element in $\mathbb{N}_{\geq 0} \times \mathbb{N}_{\geq 1}$. A q-step is said to have cost c_Q, the cost c_A is assigned to a non-q-step.[10]

[10] This generalizes the notion of von zur Gathen (1991), where only $c_Q = 0$ was considered. He assumed "[...] that *computing a qth power is for free.*", von zur Gathen (1991), p. 360.

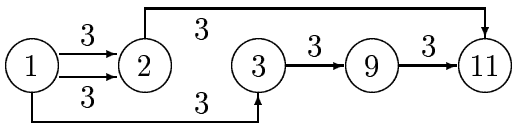
Similar to original addition chains, we assume $1 = a_0 < a_1 < \cdots < a_l$ and describe γ in terms of $\mathcal{S}(\gamma)$ and $\mathcal{R}(\gamma) = \{(a_{j(i)}, a_{k(i)})\colon 1 \le i \le l, k(i) \ne -q\} \cup \{(a_{j(i)}, -q)\colon 1 \le i \le l, k(i) = -q\}$. We set $\ell_q(\gamma) = \#\mathcal{S}(\gamma) - 1$ and

$$\ell_q(e) = \min\{\ell_q(\gamma)\colon \gamma \text{ computes } e\}. \tag{3.7}$$

Furthermore, let $Q(\gamma)$ be the number of q-steps in γ and $A(\gamma)$ be the number of non-q-steps. Then the *weighted length* $L(\gamma)$ is defined by

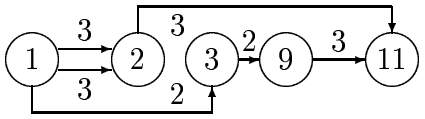
$$L(\gamma) = c_Q \cdot Q(\gamma) + c_A \cdot A(\gamma).$$

EXAMPLE 3.1 CONTINUED. In the finite field $\mathbb{F}_{3^n}$ the Frobenius automorphism raises an element to its third power. Thus, we can model this using a 3-addition chain. Here a 3-step stands for the Frobenius automorphism, a non-3-step for a multiplication. The exponent $e = 11$ has ternary representation $(11)_3 = (102)$. The 3-addition chain $\gamma = ((0,0),(0,-3),(2,-3),(3,1))$ with weight $(3,3)$ has the following graph



$$L = 3 \cdot A + 3 \cdot Q = 12$$

The graph of γ modifies as follows if we attach the weight $(2,3)$, i.e. we assume that the Frobenius automorphism can be computed faster than a multiplication by $c = \frac{2}{3}$.



$$L = 3 \cdot A + 2 \cdot Q = 10 \qquad \Diamond$$

A compiler for q-addition chains. We finally remark that a given q-addition chain γ computing $e \in \mathbb{N}_{\ge 1}$ can be easily translated into an algorithm to evaluate a power A^e in a monoid $\mathcal{G}$.

ALGORITHM 3.8. Exponentiation with q-addition chains.

Input: A monoid $\mathcal{G}$, an element $A \in \mathcal{G}$, a scalar $q \in \mathbb{N}_{\ge 2}$, an exponent $e \in \mathbb{N}_{\ge 0}$, and a q-addition chain $\gamma = ((j(1),k(1)), \ldots, (j(l),k(l)))$ computing e.

Output: The power A^e in $\mathcal{G}$.

1. If $e = 0$ then return 1.
2. For $1 \le i \le l$ do 3–4
3. If the pair $(a_{j(i)}, a_{k(i)})$ is a non-q-step such that $k(i) \ne -q$ then compute $A^{a_i} \leftarrow A^{a_{j(i)}} \cdot A^{a_{k(i)}}$.

4. Else Compute $A^{a_i} \leftarrow (A^{a_{j(i)}})^q$.
5. Return A^e.

Since we are mainly interested in exponentiation in the finite field $\mathbb{F}_{q^n}$, we set our basic cost measure to be the number of arithmetic operations in $\mathbb{F}_q$. A weight (c_Q, c_A) can be read as follows: c_Q is the number of operations in $\mathbb{F}_q$ to compute a q-th power in $\mathbb{F}_{q^n}$, and c_A is the number of arithmetic operations in $\mathbb{F}_q$ to calculate a product of two (arbitrary) elements of $\mathbb{F}_{q^n}$. The number of operations to raise an element in $\mathbb{F}_{q^n}$ to a given power $e \in \mathbb{N}_{\geq 0}$ follows directly with Algorithm 3.8.

REMARK 3.9. *Let $\mathbb{F}_{q^n}$ be the finite field with q^n elements. Let c_Q and c_A, respectively, be the number of operations in $\mathbb{F}_q$ to raise an element in $\mathbb{F}_{q^n}$ to the q-th power, and to multiply two elements in $\mathbb{F}_{q^n}$, respectively. Let γ be a q-addition chain with weight (c_Q, c_A) computing $e \operatorname{rem} (q^n - 1)$. Then an element in $\mathbb{F}_{q^n}$ can be raised to the e-th power in sequential with*

$$L(\gamma) = c_Q \cdot Q(\gamma) + c_A \cdot A(\gamma)$$

operations in $\mathbb{F}_q$.

3.2. The algorithm of Brauer. We adapt an algorithm for original addition chains described by Brauer (1939) to our extended model. This algorithmic approach—also known as the m-ary method, see e.g. Knuth (1998), p. 464, and Gordon (1998), Section 2.2—gives an upper bound on the (sequential) exponentiation problem. In the case of finite fields $\mathbb{F}_{q^n}$, Brauer's idea leads to the following statement.

COROLLARY 3.10 (Brauer 1939). *Let $\mathbb{F}_{q^n}$ be the finite field with q^n elements, $\frac{n}{\ln^2 n} > \frac{1}{\ln^2 q}$, and c_Q and c_A be the respective numbers of operations in $\mathbb{F}_q$ to evaluate a q-th power, and to multiply two elements, respectively. Then a power of an element in $\mathbb{F}_{q^n}$ can be computed with at most*

$$c_Q \cdot \left(n - 1 + \frac{n}{\log_q^2 n} \right) + c_A \cdot \frac{n}{\log_q n} (1 + o(1))$$

operations in $\mathbb{F}_q$.

Representation and Hamming weight. Let $q \geq 2$ and m be positive integers. At the core of Brauer's algorithm is the q^m-ary representation of the exponent $e \in \mathbb{N}_{\geq 1}$. The q-ary representation of e is defined as $(e)_q = (e_{\lambda-1}, \ldots, e_0)$ such that $e = \sum_{0 \leq i < \lambda} e_i q^i$ with $e_i \in \{0, \ldots, q-1\}$ for $0 \leq i < \lambda$ and $e_{\lambda-1} \geq 1$. We call $\lambda = \lambda_q(e) = \lfloor \log_q e \rfloor + 1$ the *length of the q-ary representation* of e. The integers e_i are also called the *digits* of the q-ary representation.

REMARK 3.11. *The q-ary representation of an integer $e \in \mathbb{N}_{\geq 1}$ is uniquely determined.*

Furthermore, we set $\omega_q(e) = \sum_{0 \leq i < \lambda} e_i$, the *sum of digits of e with respect to q*, and $\nu_q(e) = \#\{0 \leq i < \lambda \colon e_i \neq 0\}$ for $(e)_q = (e_{\lambda-1}, \ldots, e_0)$ the *q-ary Hamming weight* of e. For $q = 2$ we have $\nu_2(e) = \omega_2(e)$ for all $e \in \mathbb{N}_{\geq 1}$.

Brauer's idea. Brauer worked on original addition chains which are connected to the binary representation of the exponent e. He suggested to group $m \geq 1$ digits of the 2-ary representation of the exponent e, which is just a change to the 2^m-ary representation of e. This reduces the number of non-2-steps in the main stage but needs precomputation. The parameter m may be used to tune the tradeoff between both stages. We give a modified version for all scalars $q \geq 2$ which follows Brauer's main ideas. The following algorithm computes a q-addition chain.

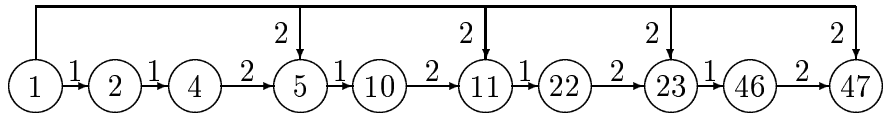
ALGORITHM 3.12. Brauer q-addition chain.

Input: An exponent $e \in \mathbb{N}_{\geq 1}$, a scalar $q \in \mathbb{N}_{\geq 2}$, and a parameter $m \in \mathbb{N}_{\geq 1}$.
Output: A q-addition chain γ computing e.

1. Set $\lambda = \lambda_{q^m}(e)$ and let e_i, $0 \leq i < \lambda$, be the digits of $(e)_{q^m} = (e_{\lambda-1}, \ldots, e_0)$.
2. Set $\mathcal{E} = \{e_{\lambda-1}, \ldots, e_0\}$ and determine a q-addition chain γ' with $\mathcal{E} \subseteq \mathcal{S}(\gamma')$.
3. Set $\gamma = \gamma'$ and $e'_{\lambda-1} = e_{\lambda-1}$.
4. For i from $\lambda - 2$ down to 0 do 5–6
5. For $1 \leq j \leq m$ do add node $e'_{i+1}q^j$ and rule $(e'_{i+1}q^{j-1}, -q)$ to γ.
6. Set $e'_i = e'_{i+1}q^m + e_i$. If $e_i \neq 0$ then add node e'_i and rule $(e'_{i+1}q^m, e_i)$ to γ.
7. Return γ.

We illustrate this algorithm by an example.

EXAMPLE 3.13. Let $e = 47$ be an exponent. We choose $q = 2$ and $m = 1$ in Algorithm 3.12 and generate the corresponding 2-addition chain of weight $(1, 2)$ for $(47)_2 = (101111)$. Since $\mathcal{E} = \{1\}$ no precomputation is necessary in step 2; we set γ' the empty addition chain with $\mathcal{S}(\gamma') = \{1\}$ and $\mathcal{R}(\gamma') = \emptyset$.



$\Diamond$

LEMMA 3.14. *The algorithm works as specified. It computes a q-addition chain γ for e with*

$$\begin{array}{rcll} Q(\gamma) & = & m \cdot (\lambda_{q^m}(e) - 1) + Q(\gamma') & \text{many } q\text{-steps, and} \\ A(\gamma) & = & \nu_{q^m}(e) - 1 + A(\gamma') & \text{non-}q\text{-steps} \end{array}$$

where γ' is a q-addition chain that computes all digits of $(e)_{q^m}$.

PROOF. We show correctness by proving the following invariant:

$$e'_i = \sum_{i \le j < \lambda} e_j (q^m)^{j-i} \in \mathcal{S}(\gamma) \text{ after lap } i \text{ of steps 4–6.}$$

Before the loop, we have $e'_{\lambda-1} = e_{\lambda-1} \in \mathcal{S}(\gamma)$ since $\gamma = \gamma'$ and $e_{\lambda-1} \in \mathcal{S}(\gamma')$ in step 3. Thus, we can suppose that the invariant holds before lap $0 \le i < \lambda - 1$. The induction hypothesis gives $e'_{i+1} = \sum_{i+1 \le j < \lambda} e_j (q^m)^{j-(i+1)} \in \mathcal{S}(\gamma)$. The nodes $e'_{i+1} \cdot q, \ldots, e'_{i+1} \cdot q^m$ are added successively to $\mathcal{S}(\gamma)$ in step 5. If $e_i = 0$ then $e'_i = e'_{i+1} q^m$ is already in $\mathcal{S}(\gamma)$ and the invariant holds. Else, e'_i is a new element of $\mathcal{S}(\gamma)$ in step 6, and we have $e'_i = e'_{i+1} q^m + e_i = \sum_{i+1 \le j < \lambda} e_j (q^m)^{j-(i+1)+1} + e_i (q^m)^{i-i} = \sum_{i \le j < \lambda} e_j (q^m)^{j-i}$. After the loop is completed, the algorithm returns γ. This q-addition chain computes $e'_0 = \sum_{0 \le j < \lambda} e_j (q^m)^{j-0} = e$ as claimed.

The number of steps can be seen as follows. In each lap of the loop, m many q-steps are added to γ. In step 6 a further non-q-step is added if and only if $e_i \neq 0$. Thus, we have a total of $\sum_{0 \le i \le \lambda-2} m = m \cdot (\lambda_{q^m}(e) - 1)$ many q-steps and $\#\{0 \le i \le \lambda - 2 \colon e_i \neq 0\} = \nu_{q^m}(e) - 1$ non-q-steps since $e_{\lambda-1} \neq 0$. Adding the steps of γ' proves the claim. □

The special case $m = 1$ and $q = 2$ chosen in Example 3.13 is well-known as the *binary addition chain* or the *binary method*, see Knuth (1962)[11]. Applied to exponentiation, it is also called *repeated squaring*. As illustrated above, the precomputation is not necessary since all digits are in $\{0, 1\}$, and 1 is given by initialization. Thus, no further elements have to be stored. The binary addition chain is a star addition chain. Since it is somehow the *basic* addition chain, we state its properties.

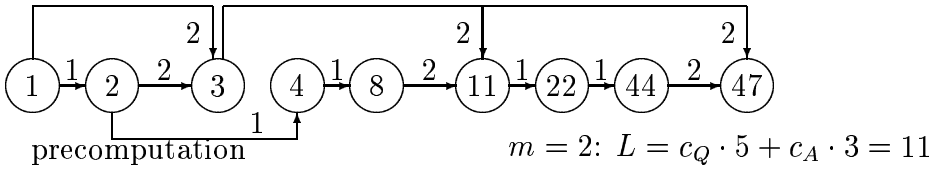
COROLLARY 3.15. *Let $e \in \mathbb{N}_{\ge 1}$ be an exponent. The* binary addition chain γ *gives the upper bound*

$$\ell_2(e) \le \ell_2^\star(e) = \lambda_2(e) - 1 + \nu_2(e) - 1 \le 2\lfloor \log_2 e \rfloor.$$

Choosing the tuning parameter. In Algorithm 3.12, we can tune the trade-off between precomputation (step 2) and the main stage (steps 4–6) with the help of the parameter m. We give an example before we discuss the details.

[11] "One method [...] which one might call the Binary Method [...].", Knuth (1962), p. 598.

EXAMPLE 3.13 CONTINUED. Let $e = 47$ be an exponent. We choose $q = 2$ and $m = 2$ in Algorithm 3.12, respectively. This generates a shorter 2-addition chain of weight $(1, 2)$ for $(47)_{2^2} = (233)$ than the one with $m = 1$ given above. In step 2 the set $\mathcal{E} = \{1, 2, 3\}$ has to be precomputed. We insert the addition chain γ' with $\mathcal{R}(\gamma') = ((1, 1), (2, 1))$ in this example.



Thus, the length is 2 units, i.e. the cost for one non-doubling or two doublings, shorter than for the binary addition chain with weight $(1, 2)$ where $m = 1$ is fixed. ◊

Now we focus on the general case. The q-addition chain γ'_1 defined by $\mathcal{S}(\gamma'_1) = \{1, 2, \ldots, q^m - 1\}$ and $\mathcal{R}(\gamma'_1) = \{(a, 1) \colon 1 \leq a < q^m - 1\}$ satisfies the specification of the precomputation. We refer to this chain as *linear addition chain*. It contains only non-q-steps, i.e. $A(\gamma'_1) = q^m - 2$ and $Q(\gamma'_1) = 0$. But we can often do slightly better by substituting the non-q-steps for multiples of q by q-steps, as remarked by Stinson (1990)[12] and von zur Gathen (1992)[13]. This idea gives a q-addition chain γ'_2 with $\mathcal{S}(\gamma'_2) = \mathcal{S}(\gamma'_1)$, but $\mathcal{R}(\gamma'_2) = \{(a - 1, 1) \colon 2 \leq a < q^m \text{ and } a \not\equiv 0 \bmod q\} \cup \{(\frac{a}{q}, -q) \colon 1 < a < q^m \text{ and } a \equiv 0 \bmod q\}$. This second q-addition chain has $Q(\gamma'_2) = \lfloor \frac{q^m - 2}{q} \rfloor = q^{m-1} - 1$ many q-steps and only $A(\gamma'_2) = q^m - 2 - Q(\gamma'_2) = q^m(1 - \frac{1}{q}) - 1$ non-q-steps.

We want to choose a suitable parameter $m \geq 1$ to obtain the lowest weighted length $L(\gamma)$. Observe that for all $e \in \mathbb{N}_{\geq 1}$, we have

$$\begin{aligned} \lambda_{q^m}(e) = \lfloor \log_{q^m} e \rfloor + 1 = \left\lfloor \frac{\log_q e}{\log_q q^m} \right\rfloor + 1 = \lfloor \frac{1}{m} \log_q e \rfloor + 1 \\ \leq \frac{1}{m} \log_q e + 1 < \frac{1}{m} \big(\lfloor \log_q e \rfloor + 1 \big) + 1 = \frac{1}{m} \lambda_q(e) + 1. \end{aligned}$$

Furthermore, we use another helpful function. Define $W \colon \mathbb{R}_{\geq 0} \to \mathbb{R}$ by $W(x) = w$ with w the solution of the equation $w \cdot \exp(w) = x$. This function is called *Lambert's W function*, following Corless *et al.* (1993)[14]. The problem to find an m that minimizes the length of a *Brauer q-addition chain* can be reduced to an equation of this type.

[12] Algorithm *compute-small-powers* in Stinson (1990), p. 713.

[13] "[...] we compute all x^d [...] with $q < d < q^r$, $d \not\equiv 0 \bmod q$, [...] the ones with $d \equiv 0 \bmod q$ can be computed free of charge.", von zur Gathen (1992), p. 82.

[14] "[...] we proposed to call this *Lambert's W function*, because Lambert set the first problem which required W for its solution, and because Euler attributed the series with which he solved the problem to Lambert.", Corless *et al.* (1993), p. 12.

THEOREM 3.16. *Let q and e be positive integers with $q \geq 2$. Then there exists a Brauer q-addition chain γ of weight (c_Q, c_A) for e of length at most*

$$L(\gamma) \leq c_A \cdot \frac{\ln q}{2}\frac{\lambda}{w} \cdot \left(1 + \frac{q^2-q}{c+q-1} \cdot \frac{1}{2w}\right) + c_Q \cdot \lambda \cdot \left(1 + \frac{q \ln q}{c+q-1}\left(\frac{1}{2w}\right)^2\right)$$

where $c = \frac{c_Q}{c_A}$, and $\lambda = \lambda_q(e)$ and $w = W\left(\frac{1}{2}\sqrt{\frac{\lambda(c+q-1)}{q}\ln q}\right)$.

PROOF. We assume $Q(\gamma') = q^{m-1} - 1$ and $A(\gamma') = q^m \cdot (1 - \frac{1}{q}) - 1$ in Algorithm 3.12 step 2. We have $\nu_{q^m}(e) \leq \lambda_q(e)$, and we substitute $\lambda_{q^m}(e)$ by $\frac{1}{m}\lambda + 1$ with $\lambda = \lambda_q(e) \geq 1$. This gives an upper bound on the length of the q-addition chain γ with weight (c_Q, c_A):

$$\begin{aligned} L(\gamma) &\leq c_Q \cdot \left(m \cdot (\frac{1}{m}\lambda + 1 - 1) + q^{m-1} - 1\right) \\ &\quad + c_A \cdot \left(\frac{1}{m}\lambda + 1 - 1 + q^m - q^{m-1} - 1\right) \\ &= c_Q \cdot \left(\lambda + q^m \cdot \frac{1}{q} - 1\right) + c_A \cdot \left(\frac{1}{m}\lambda + q^m \cdot \frac{q-1}{q} - 1\right) \qquad (3.17) \\ &= c_A \cdot \left(q^m \cdot \left(\frac{c}{q} + \frac{q-1}{q}\right) + \frac{1}{m}\lambda + c\lambda - (c+1)\right) \end{aligned}$$

where $c = \frac{c_Q}{c_A} \geq 0$. For simplicity, we set $u = \frac{c+q-1}{q} > 0$. We define the function

$$L\colon \mathbb{R}_{\geq 1} \to \mathbb{R} \text{ by } L(m) = u \cdot q^m + \frac{1}{m}\lambda + c\lambda - (c+1).$$

This is a continuous and differentiable function with

$$L'(m) = u \cdot \ln q \cdot q^m - \frac{1}{m^2}\lambda.$$

To find the extrema, we have to solve $u \ln q \cdot q^m - \frac{1}{m^2}\lambda = 0$. We can alternatively solve $u \ln q \cdot m^2 \cdot \exp(m \ln q) - \lambda = 0$. Observe that $m \geq 1$ and $q \geq 2$, which gives

$$\begin{aligned} & \ln q \cdot m^2 \exp(m \ln q) = \frac{\lambda}{u} \\ \Leftrightarrow \quad & \sqrt{\ln q \cdot m^2 \exp(m \ln q)} = \sqrt{\frac{\lambda}{u}} \\ \Leftrightarrow \quad & \left(\frac{1}{2}\ln q \cdot m\right) \cdot \exp\left(\frac{1}{2}\ln q \cdot m\right) = \frac{1}{2}\sqrt{\frac{\lambda \cdot \ln q}{u}}\,. \end{aligned}$$

We set $M = \frac{1}{2}\ln q \cdot m$. The equation

$$M \cdot \exp(M) = \frac{1}{2}\sqrt{\frac{\lambda \cdot \ln q}{u}}$$

has the solution $M = W\left(\frac{1}{2}\sqrt{\frac{\lambda \cdot \ln q}{u}}\right) \geq 0$. Therefore, the only possible extrema is

$$m = \frac{2}{\ln q} \cdot W\left(\frac{1}{2}\sqrt{\frac{\lambda \cdot \ln q}{u}}\right).$$

Observe that $L(m)'' = u \ln^2 q \cdot q^m + \frac{1}{m^3}\lambda > 0$ for all $m \geq 1$, and we have indeed a minimum. We substitute m by $\lceil m \rceil$ and set $w = W\left(\frac{1}{2}\sqrt{\frac{\lambda \cdot \ln q}{u}}\right)$. Inserting the solution in (3.17) gives

$$\begin{aligned} L(\gamma) \quad &\leq \quad c_A \cdot \left(\tfrac{\ln q}{2w} \cdot \lambda + \tfrac{q-1}{q} \cdot q^{\frac{2w}{\ln q}+1} - 1\right) \\ &\quad + c_Q \cdot \left(\tfrac{1}{q} \cdot q^{\frac{2w}{\ln q}+1} + \lambda - 1\right). \end{aligned} \tag{3.18}$$

We have $q^{2w/\ln q} = \exp(2w)$, and by definition $W(x) \cdot \exp(W(x)) = x$, which gives

$$\exp(2w) = \exp(w)^2 = \frac{1}{4w^2} \frac{q\lambda \ln q}{c + q - 1}.$$

Inserting this in (3.18) and sorting the terms proves the claim. □

In the literature one finds another choice for m which avoids Lambert's W function. For original addition chains with $q = 2$ and $c_Q = c_A = 1$, Gordon (1998) suggests $m = \log_2\log_2 e - 2\log_2\log_2\log_2 e$. A slightly different choice for m has been given for the original case by Brauer (1939). We prove Corollary 3.10 applying the first choice.

Proof (of Corollary 3.10). In the finite field $\mathbb{F}_{q^n}$, we have $A^e = A^{e'}$ with $e' = e \operatorname{rem} (q^n - 1)$ due to Fermat's Little Theorem 2.3. Thus, we assume the exponent e to be less than q^n, i.e. $\lambda_q(e) \leq n$. We choose $m = \lfloor \log_q n - 2\log_q\log_q n \rfloor + 1 \geq 1$ and set $c = \frac{c_Q}{c_A}$. We insert this choice of m—which is non-zero since $\frac{n}{\ln^2 n} \geq \frac{1}{\ln^2 q}$—in (3.17):

$$\begin{aligned} L(\gamma) \quad &\leq \quad c_Q \cdot \left(n - 1 + q^{\lfloor \log_q n - 2\log_q\log_q n \rfloor + 1} \cdot \frac{1}{q}\right) \\ &\quad + c_A \cdot \left(q^{\lfloor \log_q n - 2\log_q\log_q n \rfloor + 1} \cdot \frac{q-1}{q} + \frac{n}{\lfloor \log_q n - 2\log_q\log_q n \rfloor + 1} - 1\right) \end{aligned}$$

$$\begin{aligned}
&\leq c_A \cdot \left((q-1) \cdot \frac{n}{\log_q^2 n} + \frac{n}{\log_q n - 2\log_q \log_q n} \right) + c_Q \cdot \left(n - 1 + \frac{n}{\log_q^2 n} \right) \\
&= c_A \cdot \frac{n}{\log_q n} \cdot \left(\frac{q-1}{\log_q n} + \frac{\log_q n}{\log_q n - 2\log_q \log_q n} \right) + c_Q \cdot \left(n - 1 + \frac{n}{\log_q^2 n} \right) \\
&= c_A \cdot \frac{n}{\log_q n} \cdot \left(1 + \frac{2\log_q \log_q n}{\log_q n - 2\log_q \log_q n} + \frac{q-1}{\log_q n} \right) \\
&\quad + c_Q \cdot \left(n - 1 + \frac{n}{\log_q^2 n} \right)
\end{aligned}$$

But we have $\lim_{n\to\infty} \frac{2\log_q \log_q n}{\log_q n - 2\log_q \log_q n} = 0$ and also $\lim_{n\to\infty} \frac{q-1}{\log_q n} = 0$ and the claim follows. □

We finally remark that an analogous choice $m = \lfloor \lambda_2(e) - 2\log_2 \lambda_2(e) \rfloor + 1$ results in the following upper bound on the length of the shortest (original) addition chain for $e \in \mathbb{N}_{\geq 3}$:

$$\ell_2(e) \leq \lambda_2(e) + \frac{\lambda_2(e)}{\log_2 \lambda_2(e)}(1 + o(1)). \tag{3.19}$$

Erdös (1960) proved that this upper bound is the best possible. In fact, he proved the following result.

FACT 3.20 (Erdös 1960). *For almost all $e \in \mathbb{N}_{\geq 1}$ (i.e. for all e expect a sequence of density 0)*

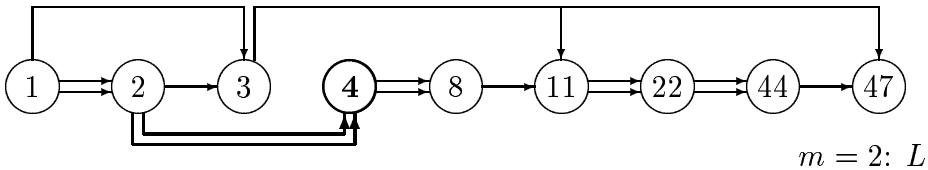
$$\ell_2(e) = \log_2 e + \frac{\log_2 e}{\log_2 \ln e}(1 + o(1)).$$

Knuth (1998), p. 470, conjectured that $\ell_2(e) \geq \lambda_2(e) + \log_2 \omega_2(e)$ is a lower bound that holds for all $e \in \mathbb{N}_{\geq 1}$. The best proven lower bound is due to Schönhage (1975), and this estimate is very close to the conjecture; he showed $\ell_2(e) \geq \log_2 e + \log_2 \omega_2(e) - 2.13$.

Brauer for star addition chains. In the last paragraph of Section 3.2 we focus on the special case $q = 2$ and star addition chains. Obviously, the *binary addition chain* is a star addition chain, while *Brauer addition chains* do not always have this property, as shown in Example 3.21. But Brauer's idea can be modified to generate star addition chains in the case $q = 2$. In fact, only one doubling has to be substituted by a non-doubling.

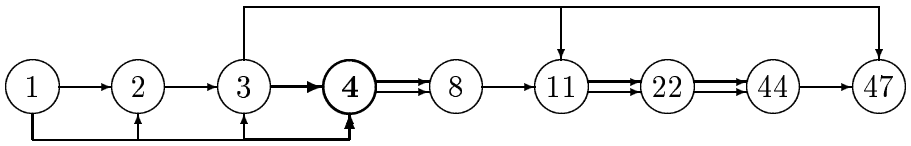
EXAMPLE 3.21. Let $m = 2$ and $e = 47$ with $(e)_{2^2} = (233)$ as before. The graph of this Brauer (original) addition chain for 47 is drawn below[15]. Obviously, it is no star addition chain.

[15] We do not label the edges in the case of original addition chains.



$m = 2$: $L = 8$

We can alternatively substitute the doubling that generates $2e_{\lambda-1}$ (here: $2 \cdot 2 = 4$) by a non-doubling since $2e_{\lambda-1} \leq 2 \cdot (2^m - 1)$. Furthermore, we precompute the set $\mathcal{E} = \{1, \ldots, 2^m - 1\}$ by the linear addition chain.



$m = 2$: $L = 8$ $\Diamond$

This algorithmic idea has already been described by Knuth (1998) in his proof of a result of Brauer (1939). The modified algorithm generates a star addition chain for given positive integers m and e of length

$$L(\gamma) \leq m \cdot (\lambda_2(e) - 1) + \nu_{2^m}(e) + 2^m - 3.$$

This is the same estimate as in Lemma 3.14 when the digits of e are precomputed by a linear addition chain. The only modification is the exchange of doubling steps by non-doubling steps.

COROLLARY 3.22 (Knuth 1998, Section 4.6.3, Theorem D). *For a positive integer e there is a star addition chain γ computing e in length at most*

$$\ell_2^\star(e) \leq L(\gamma) \leq \lambda_2(e) + \frac{\lambda_2(e)}{\log_2 \lambda_2(e)}(1 + o(1)).$$

Thus, both $\ell_2(e)$ and $\ell_2^\star(e)$ show the same asymptotic behavior for $e \to \infty$. Obviously, $\ell_2(e) \leq \ell_2^\star(e)$ for all $e \in \mathbb{N}_{\geq 1}$. But equality does not always hold. Knuth (1998) reports on calculations[16] that show that $e = 12509$ is the smallest counter-example. Hansen (1959), Satz 1 showed that the difference $\ell_2^\star(e) - \ell_2(e)$ can become arbitrary large for suitable exponents.

We will use star addition chains which are shorter than the binary addition chain in Section 6.5. They will be useful to decrease the time to compute the inverse of an element in $\mathbb{F}_{q^n}^\times$.

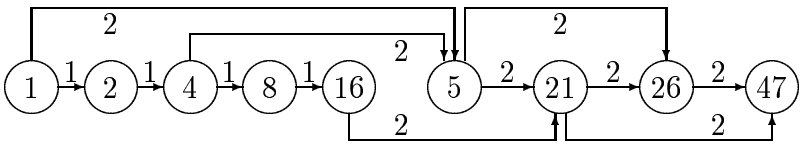
[16] "Extensive computer calculations have shown that $n = 12509$ is the smallest value with $l(n) < l^*(n)$.", Knuth (1998), p. 477. [Here $l(n)$ and $l^*(n)$ are our $\ell_2(n)$ and $\ell_2^\star(n)$, respectively.]

3.3. Addition chains for sets. Up to now we have used a q-addition chain γ' in Algorithm 3.12 step 2 which contains all possible non-zero digits $\{1, 2, \ldots, q^m - 1\}$ of the q^m-ary representation $(e)_{q^m}$. But we only have to find a q-addition chain that computes the set $\mathcal{E} = \{e_0, \ldots, e_{\lambda-1}\}$ of all non-zero digits of $(e)_{q^m}$. This idea and respective experiments on it can already be found in Bos & Coster (1990)[17]. A q-addition chain γ computes a finite set $\mathcal{E} \subset \mathbb{N}_{\geq 1}$ if $\mathcal{E} \subseteq \mathcal{S}(\gamma)$. Yao (1976) presented a construction to generate an original addition chain that computes $\mathcal{E} \subset \mathbb{N}_{\geq 1}$. Brickell *et al.* (1993) used a similar approach to generate an addition chain for a single exponent $e \in \mathbb{N}_{\geq 1}$. We refer to such a chain as *BGMW addition chain*. We present their ideas but extend them to weighted q-addition chains.

BGMW addition chains. Algorithm 3.12 computes and stores the digits $e_0, \ldots, e_{\lambda-1}$ of the q^m-ary representation of a given exponent e in the precomputation stage. The idea of Brickell *et al.* (1993) is to precompute all positions $(q^m)^0, \ldots, (q^m)^{\lambda-1}$ of $(e)_{q^m}$ in place of all digits. These elements are used to calculate e by grouping them according to the digits $e_0, \ldots, e_{\lambda-1}$:

$$e = \sum_{0 \leq i < \lambda} e_i (q^m)^i = \sum_{0 \leq j < q^m} \left(\sum_{\substack{0 \leq i < \lambda \\ e_i \geq j}} q^{mi} \right). \tag{3.23}$$

EXAMPLE 3.24. For the exponent $e = 47$ and the scalar $q = 2$ we choose the parameter $m = 2$. Then the BGMW addition chain with weight $(1, 2)$ for $(47)_2 = (10|11|11)$ is given by the graph:



precomputation $\qquad m = 2\text{: } L = c_Q \cdot 4 + c_A \cdot 4 = 12$ $\qquad \Diamond$

ALGORITHM 3.25. BGMW addition chain.

Input: An exponent $e \in \mathbb{N}_{\geq 1}$, a scalar $q \in \mathbb{N}_{\geq 2}$, and a parameter $m \in \mathbb{N}_{\geq 1}$.
Output: A q-addition chain γ for e.

1. Set $\lambda = \lambda_{q^m}(e)$ and let $(e)_{q^m} = (e_{\lambda-1}, \ldots, e_0)$ be the q^m-ary representation of e. Set γ the empty addition chain with $\mathcal{S}(\gamma) = \{1\}$.
2. For $1 \leq i < \lambda$ do

[17] "Of course, it is more efficient to use an addition sequence producing only the needed numbers.", Bos & Coster (1990), p. 403.

3. For $1 \le j \le m$ do add node $q^{(i-1)m+j}$ and rule $(q^{(i-1)m+(j-1)}, -q)$ to γ.
4. Set $a = 0$ and $b = 0$.
5. For j from $q^m - 1$ down to 1 do 6–8
6. For all $e_i = j$ do
7. Set $a \leftarrow a + q^{mi}$. If $a \notin \mathcal{S}(\gamma)$ then add node a and rule $(a - q^{mi}, q^{mi})$ to γ.
8. Set $b \leftarrow b + a$. If $b \notin \mathcal{S}(\gamma)$ then add node b and rule $(b - a, a)$ to γ.
9. Return γ.

LEMMA 3.26. *The algorithm computes a q-addition chain for e with*

$$\begin{array}{rcll} Q(\gamma) & = & m \cdot (\lambda_{q^m}(e) - 1) & \text{many } q\text{-steps, and} \\ A(\gamma) & \le & \nu_{q^m}(e) + q^m - 3 & \text{non-}q\text{-steps} \end{array}$$

and all q-steps and no non-q-steps are performed in the precomputation stage.

PROOF. Correctness follows with (3.23). All q-steps are performed in steps 2–3. There are $Q(\gamma) = (\lambda - 1) \cdot m$ many q-steps. The main stage (steps 4–8) contains only non-q-steps. In the worst case, the outer loop adds a node to γ in each turn except the first one since b is initially 0. This yields $q^m - 1 - 1$ non-q-steps. For the inner loop we have $\sum_{1 \le j < q^m} (\#\{0 \le i < \lambda \colon e_i = j\}) = \#\{0 \le i < \lambda \colon e_i \ne 0\} = \nu_{q^m}(e)$ non-q-steps. Since we have $a = 0$ before entering the loop, this number can be decreased by 1. Thus, we count a total of at most $A(\gamma) \le q^m - 2 + \nu_{q^m}(e) - 1$ non-q-steps as claimed. □

The precomputed multiples of q do only depend on the number of digits in $(e)_{q^m}$ but not on the specific digits. Therefore, we need to execute the precomputation stage only once if we want to compute a q-addition chain for a finite set $\mathcal{E} = \{e^{(1)}, \ldots, e^{(t)}\} \subset \mathbb{N}_{\ge 1}$ with $t \ge 1$. Originally, Yao (1976) has used this observation as key argument of his proof. It is the basic idea of the following corollary.

COROLLARY 3.27 (Yao 1976). *Let $\mathcal{E} = \{e^{(1)}, \ldots, e^{(t)}\}$ be a finite set of positive integers and $e = \max \mathcal{E}$. There is a q-addition chain γ with weight (c_Q, c_A) computing $\mathcal{E}$ with length at most*

$$L(\gamma) \le c_Q \cdot \lambda_q(e) + c_A \cdot \sum_{1 \le i \le t} \frac{\lambda_q(e^{(i)})}{\log_q \lambda_q(e^{(i)})} \, (1 + o(1)) \, .$$

PROOF. We have $\lambda_q(e^{(i)}) \le \lambda_q(e)$ for all $1 \le i \le t$. Thus, the precomputation stage for $e = \max \mathcal{E}$ covers all elements which are necessary in the main stage to calculate all $e' \in \mathcal{E}$. Since $m \cdot (\lambda_{q^m}(e) - 1) \le \lambda_q(e)$, the number of q-steps may be assumed to be independent of the tuning parameter m.

For an element $e' \in \mathcal{E}$, we have $\nu_{q^m}(e') - 1 \leq \lambda_{q^m}(e') - 1 \leq \frac{1}{m}\lambda'$ with $\lambda' = \lambda_q(e')$. This gives a total sum of at most $\frac{1}{m}\lambda' + q^m - 2$ non-q-steps when performing steps 4–8 in Algorithm 3.25 for e'. We choose the tuning parameter m for each element of $\mathcal{E}$ separately. If $\frac{\lambda'}{\log_q^2 \lambda'} < 1$, we set $m = 1$. Otherwise, let $m = \lfloor \log_q \lambda' - 2\log_q \log_q \lambda' \rfloor + 1$. Then there are

$$\begin{aligned} & \frac{\lambda'}{\lfloor \log_q \lambda' - \log_q \log_q \lambda' \rfloor + 1} + q^{\lfloor \log_q \lambda' - \log_q \log_q \lambda' \rfloor + 1} - 2 \\ \leq\ & \frac{\lambda'}{\log_q \lambda' - \log_q \log_q \lambda'} + q \cdot \frac{\lambda'}{\log_q^2 \lambda'} \\ =\ & \frac{\lambda'}{\log_q \lambda'} \left(1 + \frac{2\log_q \log_q \lambda'}{\log_q \lambda' - 2\log_q \log_q \lambda'} + \frac{q}{\log_q \lambda'} \right) = \frac{\lambda'}{\log_q \lambda'} \left(1 + o(1)\right). \end{aligned}$$

many non-q-steps. Summing up the number of non-q-steps for all elements in $\mathcal{E}$ proves the claim. □

For a finite field $\mathbb{F}_{q^n}$, we can assume $0 < e < q^n$ and $\lambda_q(e) \leq n$ by Fermat's Little Theorem 2.3. Choosing $m = \lfloor \log_q n - 2\log_q \log_q n \rfloor + 1$ in Algorithm 3.25 gives the following upper bound on exponentiation in finite fields.[18]

COROLLARY 3.28 (Brickell *et al.* 1993). *Let $\mathbb{F}_{q^n}$ be the finite field with q^n elements, $\frac{n}{\log_q^2 n} \geq 1$, and $c_Q \in \mathbb{N}_{\geq 0}$ and $c_A \in \mathbb{N}_{\geq 1}$ be the number of operations in $\mathbb{F}_q$ to calculate a q-th power, and to multiply two elements, respectively. Then a power of an element in $\mathbb{F}_{q^n}$ can be computed with at most*

$$c_Q \cdot n + c_A \cdot \frac{n}{\log_q n} \left(1 + o(1)\right)$$

operations in $\mathbb{F}_q$.

Therefore, applying Algorithm 3.25 as the precomputation stage in Algorithm 3.12 might reduce the number of steps to evaluate a given power. In fact, if the computation of a (suitable) power of q is (much) faster than the successively calculation by q-steps, this strategy may be more powerful than one of the above algorithms. An example for this had been given by Gao *et al.* (2000), Algorithm 2.3.

3.4. Experiments. As a first step stone of our experiments, we implemented both Algorithm 3.12 (for *binary addition chain* and *Brauer addition chain*) and Algorithm 3.25 (*BGMW addition chain*) for $q = 2$ in `C++`. We also implemented a class in `C++` to handle exponents in an object-oriented fashion. The addition chain algorithms possess an interface to specify the underlying group. The arithmetic

[18] In Brickell *et al.* (1993), p. 206, one can find a similar estimate for exponents of fixed size. This is the case for finite fields.

of the group includes the two basic operations multiplication (non-doubling) and squarings (doublings). For the first experiments we do not specify the group. Here, we only have counted the number of doublings and non-doublings of the different addition chain algorithms. We use the normalized weighted length $L(\gamma)/c_A = A(\gamma) + c \cdot Q(\gamma)$ for a weight (c_Q, c_A) and $c = \frac{c_Q}{c_A}$ as our measure for quality; the shorter a 2-addition chain, the better.

One can find an extensive comparison between five different addition chain algorithms, including the three addition chains which are subject of our experiments, in von zur Gathen & Nöcker (1997, 2000). In what follows, we discuss the influence of different weights (c_Q, c_A) on the different addition chains. In von zur Gathen & Nöcker (1997), the weight was fixed, and the test series for the addition chain algorithm differed by their Hamming weight $\nu_2(e)$. The binary length of the exponents was restricted to $\lambda_2(e) \in \{160, 512, 1024\}$.

We selected 50 values $n \approx 200i$ with $1 \leq i \leq 50$ for our experiments. For each value of n, we chose 10000 exponents at random with binary length $\lambda_2(e) = n$. For each exponent, we generated the binary addition chain, the Brauer addition chain, and the BGMW addition chain for five different sets of weights, see Tables A.1–A.5 in Appendix A. For each n of this 15 test series, we determined the average number of doublings (A) and non-doublings (Q) for all 10000 trials. Furthermore, we computed the normalized length $L = A + c \cdot Q$ with $c = \frac{c_Q}{c_A}$ for each n and weight (c_Q, c_A) to facilitate comparison.

We chose five different sets of weight: original addition chains are given by weight $(1, 1)$ (see Figure 3.1 and Table A.1); 2-addition chains with free doublings are modeled by weight $(0, 1)$ (see Figure 3.3 and Table A.5). The remaining three series were inspired by constructions of the finite field $\mathbb{F}_{2^n}$ using different polynomial basis representations. For one series documented in Figure 3.2 and Table A.2, we have a weight close to $(2, 3)$, see the results of Section 5.2.1. For the two other series the weight decreases for increasing n; the quotient $c = \frac{c_Q}{c_A}$ starts approximately at $\frac{1}{2}$ and is close to 0 for $n \approx 10000$, see Table A.3 and Table A.4, respectively. This is the situation that we will discussed in Section 5.2.2 and Section 5.2.3, respectively.

For Brauer addition chains and BGMW addition chains we had to select a tuning parameter m. We chose that m which generates the minimal average normalized length L over all 10000 trials for fixed n and c. Our experiments validate the theoretical results of Theorem 3.16 on m (see column 6 of Tables A.1–A.5). In particular, the optimal tuning parameter m for Brauer addition chains depends not only on n but is also influenced by the ratio $c = \frac{c_Q}{c_A}$!

The experiments showed the worst performance for the binary addition chain, independent from the binary length n and c. This meets our theoretic estimates. The speed-up between the binary addition chain and the other two chains depends on the weight. For both the Brauer and the BGMW addition chain the speed-up is roughly 1.3 if $c_Q = c_A = 1$, see Table A.1, columns 10 and 15. The best speed-

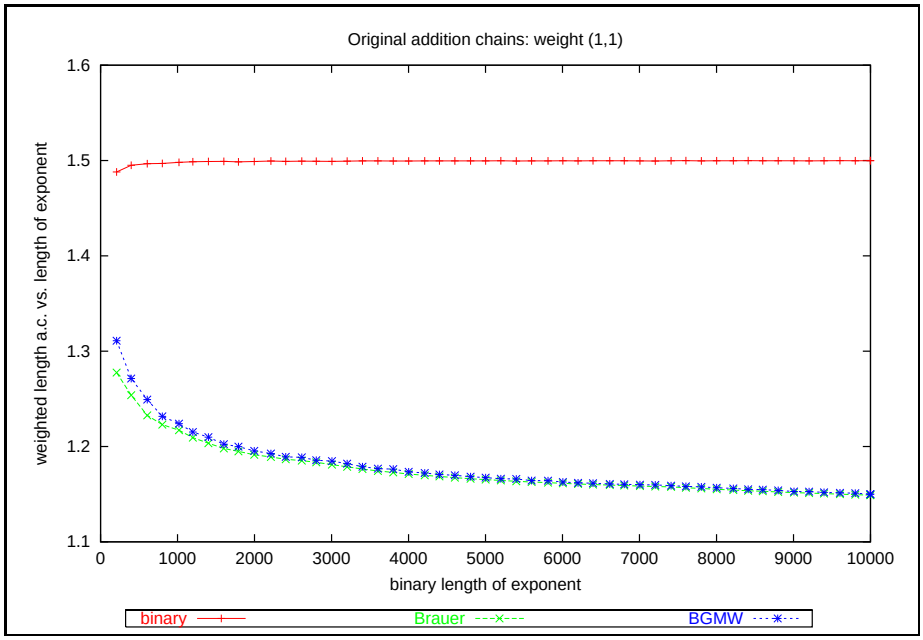


Figure 3.1: Comparison between three different addition chain algorithms for original addition chains: *binary addition chain* ($+$), *Brauer addition chain* ($\times$), and *BGMW addition chain* ($*$). The figure shows the quotient between the weighted length $L = c_A \cdot A + c_Q \cdot Q$ and the binary length of the exponent $\lambda_2(e) = n$ multiplied with c_A. On the x-axis the binary length of the exponent is marked, the quotient is given on the y-axis.

up in the experiment was achieved for $c_Q = 0$. Then Brauer addition chains had a normalized length which is only 27% of the length of the binary addition chain for $n = 9998$, see Table A.5, column 10.

Our experiments as documented in Figure 3.1 are also in agreement with the theory for original addition chains if we compare Brauer and BGMW addition chains. Here, both algorithms generated 2-addition chains of nearly the same normalized length L, see also Table A.1, columns 9 and 14. Corollary 3.10 and Lemma 3.26 give the same estimates in this case. But Figure 3.3 shows a significant difference between both methods if $c_Q = 0$. The average length of a Brauer addition chain for $n = 9998$ is about 10% shorter than that of the BGMW addition chain. Nevertheless, this coincides with the theoretical results if we compare the bound given in Lemma 3.14 to that in Lemma 3.26. We have

$$c_Q \cdot \left(m \cdot (\lambda_{q^m}(e) - 1) + q^{m-1} - 1\right) + c_A \cdot \left(\nu_{q^m}(e) + q^m - q^{m-1} - 2\right)$$

for a Brauer q-addition chain versus

$$c_Q \cdot (m \cdot (\lambda_{q^m}(e) - 1)) + c_A \cdot (\nu_{q^m}(e) + q^m - 3)$$

for a BGMW q-addition chain. Thus, the latter has $q^{m-1} - 1$ less q-steps, but the number of non-q-steps increases by the same order. We conclude that Brauer

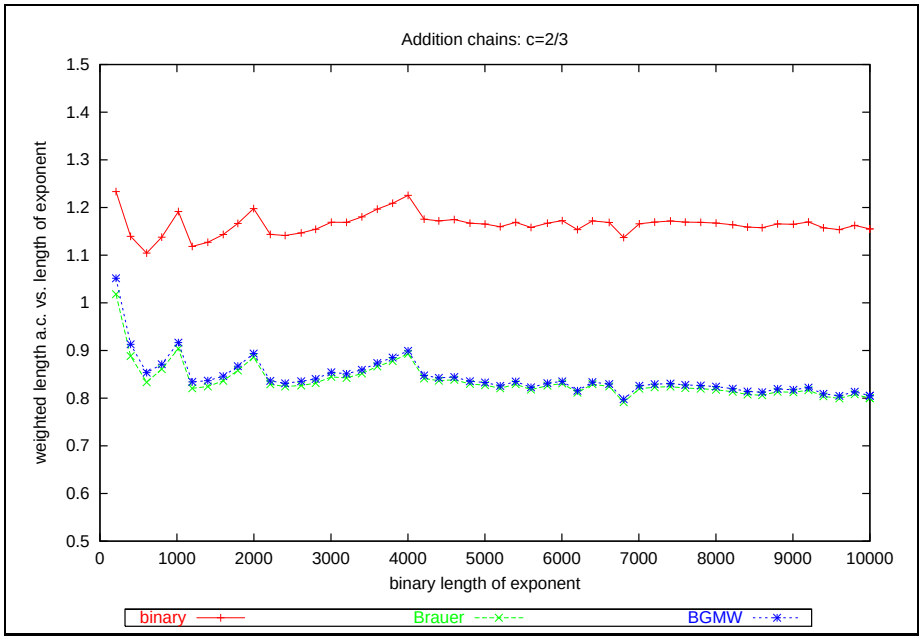


Figure 3.2: The same experiment as illustrated in Figure 3.1 but for weight roughly $(2, 3)$. This situation is given for a polynomial basis representation of $\mathbb{F}_{2^n}$ with arbitrary modulus and division by remainder via Newton inversion, see Section 5.2.1.

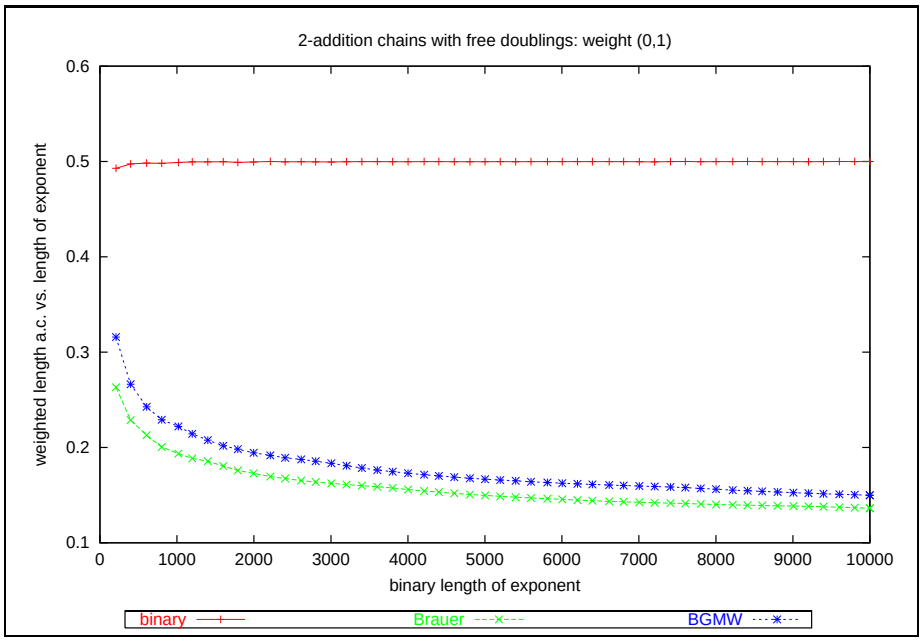


Figure 3.3: The same experiment as illustrated in Figure 3.1 but for weight $(0, 1)$, i.e. doublings are free.

q-addition chain have the advantage of generating a shorter overall q-addition chain if c_Q is less than c_A. Our experiments validated this effect for weighted 2-addition chains.

Therefore, we will chose Brauer 2-addition chains to generate the sequential exponentiation algorithm since it always computes shorter addition chains if $c_Q \ll c_A$. In the experiments described in the subsequent sections, we connect the implementation of Algorithm 3.12 with different routines for multiplications and squaring via the interface mentioned above.

4. Parallel exponentiation

How fast can we evaluate a power in parallel if the number of processors is not a bottleneck? Surprisingly, we have found more than one answer in the literature. The answer given by Kung (1976) and Borodin & Munro (1975) holds for a model of *unitary cost*. Using our language their result is applicable to original addition chains. A model of *addition chains with free scalar* has been described by von zur Gathen (1991)[19]. He allows q-steps without cost for a fixed scalar $q \in \mathbb{N}_{\geq 2}$, i.e. $c_Q = 0$, and assumes that non-q-steps have unitary cost $c_A = 1$. Both models are covered by our approach of weighted q-addition chains.

This section presents the first main result of our work. We have already identified the weight (c_Q, c_A) to be a helpful tool for the construction of sequential q-addition chains. In this section the ratio $c = \frac{c_Q}{c_A}$ determines how successful a q-addition chain can be computed in parallel. We first develop a parallel algorithm for q-addition chains that depends on c. The starting point is an algorithm of Borodin & Munro (1975) (Section 4.1). In the second part of this section, we generalize an idea of Kung (1976) to give a lower bound on the depth of a weighted q-addition chain for an exponent $e \in \mathbb{N}_{\geq 1}$. This bound also depends on c. The main tool is a so-called *scaling for* $(e)_q$ *with respect to* c. It depends not only on e but also on the scalar $q \in \mathbb{N}_{\geq 2}$ and the quotient $c = \frac{c_Q}{c_A} \in \mathbb{Q}_{\geq 0}$ (Section 4.2). Our results, both the algorithmic approach and the lower bound, close a gap by connecting works of Kung (1976) and Borodin & Munro (1975) with a result of von zur Gathen (1991) on parallel exponentiation. Our new approach points to c as an indicator on the potential speed-up which can be achieved for a given data structure (Section 4.3).

4.1. An algorithm for parallel exponentiation. In the first part of this section we answer the question by giving an algorithm (and thus an upper bound) for parallel exponentiation in our model of weighted q-addition chains. Adding the result of von zur Gathen (1991), Theorem 4.3, for weight $(0, 1)$ the algorithm gives the following upper bound on the number of operations in $\mathbb{F}_q$.

COROLLARY 4.1. *Let $\mathbb{F}_{q^n}$ be the finite field with q^n elements, and c_Q and c_A be the number of operations in $\mathbb{F}_q$ to evaluate a qth power, and to multiply two elements, respectively. Then a power of an element of $\mathbb{F}_{q^n}$ can be computed in parallel with at most*

(i) $c_Q \cdot (n-1) + c_A \cdot (\lceil \log_2(q-1) \rceil + 1)$ *operations in $\mathbb{F}_q$ if $c_Q \geq c_A \geq 1$. This can be performed with q processors.*

[19] "A very simple type of arithmetic circuit consists just of multiplications and qth powers [...]. This type of circuits corresponds to *addition chains* [...]. Free qth powers in the circuit correspond to *free multiplications by q* in the addition chain. These chains [...] will be our model for the remainder of the paper.", von zur Gathen (1991), p. 371–372.

(ii) $c_Q \cdot (n-1) + c_A \cdot \left(\lceil \log_2(q-1) \rceil + 1 + \left\lceil \log_2 \left(\min \left\{ n, \left\lceil \frac{c_A}{c_Q} \right\rceil \right\} \right) \right\rceil \right)$ *operations in* $\mathbb{F}_q$ *if* $c_A > c_Q \geq 1$. *This can be performed with* $q - 1 + \min \left\{ \left\lceil \frac{c_A}{c_Q} \right\rceil, n \right\}$ *processors.*

(iii) $c_A \cdot (\lceil \log_2(q-1) \rceil + \lceil \log_2 n \rceil)$ *operations in* $\mathbb{F}_q$ *if* $c_A \geq 1$ *and* $c_Q = 0$. *This can be performed with* $\lfloor \frac{1}{2} n(q-1)) \rfloor$ *processors.*

4.1.1. The depth of an addition chain with scalar. In this section we assume that the number of available processors is not a priori bounded. Borodin & Munro (1975) called this *unbounded parallelism.* We use the *parallel random access machine (PRAM)* as our basic model for parallel computation. The PRAM is a shared memory model with $P \in \mathbb{N}_{\geq 1}$ processors, see Figure 4.1. The P

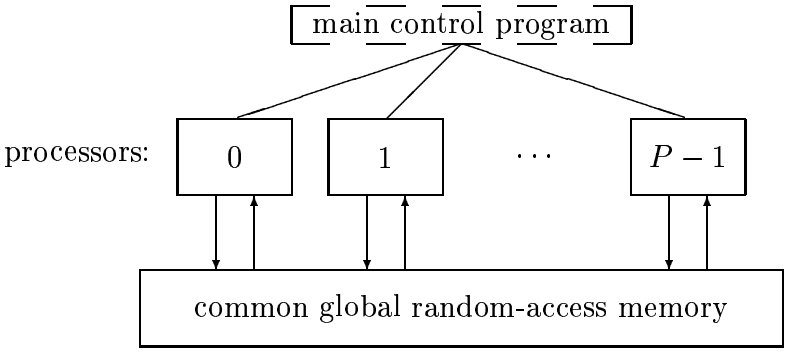


Figure 4.1: The scheme of a PRAM, see Gibbons & Rytter (1988), Figure 1.1.

processors are subscripted from 0 to $P-1$. They work synchronously and execute the same program. Each of them may work on different data. The instructions of the program can depend on the processor label. The processors communicate via the shared memory. We assume that any number of processors can simultaneously read the same data, but no two processors may write at the same time into the same memory location. For a detailed introduction on this model of parallel computing we refer to Gibbons & Rytter (1988), Section 1.1. The PRAM leads to the following rule for the parallel computation of addition chains: All previously calculated nodes of a q-addition chain may be used by any of the processors without further cost. In particular, two or more of them may simultaneously access to the same already calculated node. Thus, it is of no importance which processor has computed the predecessors of a node. This is the common idea of shared memory which neglects communication cost.

Let γ be a q-addition chain with weight (c_Q, c_A). We set

$$d_\gamma^{(c_Q,c_A)}(a) = \begin{cases} 0 & \text{if } a = 1, \\ c_Q + d_\gamma^{(c_Q,c_A)}(a/q) & \text{if } (a/q, -q) \in \mathcal{R}(\gamma), \\ c_A + \max\{d_\gamma^{(c_Q,c_A)}(a'), d_\gamma^{(c_Q,c_A)}(a-a')\} & \text{if } (a', a-a') \in \mathcal{R}(\gamma) \end{cases}$$

since we assume that each node is computed only once, and due to the shared memory it is available by all processors. We define the depth $\delta_{(c_Q,c_A)}(\gamma)$ of a q-addition chain γ with weight (c_Q, c_A) as

$$\delta_{(c_Q,c_A)}(\gamma) = \max\{d_\gamma^{(c_Q,c_A)}(a)\colon a \in \mathcal{S}(\gamma)\}.$$

The *minimal depth* for an exponent $e \in \mathbb{N}_{\geq 1}$ depends on the scalar $q \in \mathbb{N}_{\geq 2}$ and on the weight (c_Q, c_A):

$$\delta_{q,(c_Q,c_A)}(e) = \min\left\{ \begin{array}{l} \delta_{(c_Q,c_A)}(\gamma)\colon \gamma \text{ is a } q\text{-addition chain with} \\ \text{weight } (c_Q, c_A) \text{ that computes } e \end{array} \right\}. \tag{4.2}$$

We say that a q-addition chain γ with weight (c_Q, c_A) can be *computed in parallel (in depth $\delta_{(c_Q,c_A)}(\gamma)$) with $P \in \mathbb{N}_{\geq 1}$ processors* if

$$P \geq \max_{0 \leq t \leq \delta_{(c_Q,c_A)}(\gamma)} \#\{d_{\gamma,(c_Q,c_A)}(a) = t\colon a \in \mathcal{S}(\gamma)\}.$$

4.1.2. Basic algorithms. We start with original addition chains, i.e. $q = 2$ and $c_Q = c_A = 1$. The basic algorithm was given by Borodin & Munro (1975), Chapter 6, and it is illustrated in Figure 4.2. Chiou (1993) described the same algorithm for a parallel implementation of the RSA-cryptosystem[20].

ALGORITHM 4.3. Parallel exponentiation in case of unitary cost.

Input: An exponent $e \in \mathbb{N}_{\geq 1}$ with binary representation $(e)_2 = (e_{\lambda-1}, \ldots, e_0)$.
Output: An (original) addition chain γ computing e.

1. Set γ the empty addition chain with $\mathcal{S}(\gamma) = \{1\}$.
2. For all processors $0 \leq p < P = 2$ in parallel do 3–5
3. For $1 \leq i \leq \lceil \log_2 e \rceil$ do 4–5
4. If $p = 0$ and $i < \lambda_2(e)$ then add node 2^i and rule $(2^{i-1}, 2^{i-1})$ to γ.
5. If $p = 1$ and $\sum_{0 \leq j < i} e_j 2^j > 2^{i-1}$ then add node $\sum_{0 \leq j < i} e_j 2^j$ and rule $(\sum_{0 \leq j < i-1} e_j 2^j, 2^{i-1})$ to γ.
6. Return γ.

THEOREM 4.4 (Borodin & Munro 1975, Lemma 6.1.1). *Let $e \in \mathbb{N}_{\geq 1}$ be an exponent. Then there is an addition chain γ computing e in depth*

$$\delta_{2,(1,1)}(e) \leq \delta_{(1,1)}(\gamma) = \lceil \log_2 e \rceil .$$

The addition chain can be performed with 2 processors.

[20] The security of the RSA-cryptosystem is not based on the discrete logarithm problem but on the difficulty to factor large integers. It has been developed by Rivest *et al.* (1978). Their idea introduced a second family of public key cryptosystems.

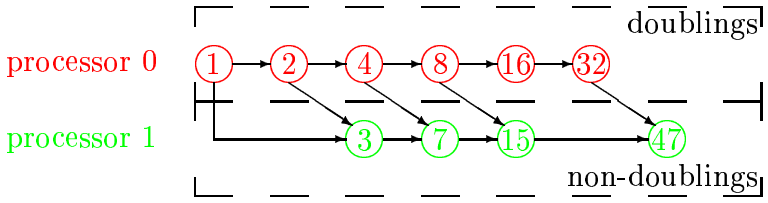


Figure 4.2: An illustration of Algorithm 4.3 for the exponent $(47)_2 = (101111)$.

PROOF. The termination of the algorithm and the number of processors are clear. We prove partial correctness by showing that the following invariant holds for steps 3–5: After lap $1 \leq i < \lambda_2(e)$ the nodes 2^i and $\sum_{0 \leq j < i} e_j 2^j$ if non-zero are in $\mathcal{S}(\gamma)$. By induction on i: Before the first lap of the loop, the invariant holds since $2^0 = 1$ is in $\mathcal{S}(\gamma)$ for the empty addition chain γ and $\sum_{0 \leq j < 1} e_j 2^j = 0$. Thus, suppose the invariant holds for $i-1$. Since $2^{i-1} \in \mathcal{S}(\gamma)$, processor 0 adds the node 2^i to $\mathcal{S}(\gamma)$. If $\sum_{0 \leq j < i} e_j 2^j \leq 2^{i-1}$ then processor 1 has to add no new node in step 5. Therefore, we assume $\sum_{0 \leq j < i} e_j 2^j > 2^{i-1}$. By the induction hypothesis both $\sum_{0 \leq j < i-1} e_j 2^j$ and 2^{i-1} are in $\mathcal{S}(\gamma)$. But $\sum_{0 \leq j < i} e_j 2^j > 2^{i-1}$ if and only if $e_{i-1} = 1$, and hence, $\sum_{0 \leq j < i} e_j 2^j$ is added to $\mathcal{S}(\gamma)$.

If $e = 2^{\lambda_2(e)-1}$ then processor 0 computes the node e in lap $i = \lambda_2(e) - 1 = \lceil \log_2 e \rceil$. If e is not a power of 2 then processor 1 adds $2^{\lambda_2(e)-1} + \sum_{0 \leq j < \lambda_2(e)-1} e_j 2^j$ to $\mathcal{S}(\gamma)$ in the final lap since $\lceil \log_2 e \rceil = \lambda_2(e)$ in this case.

Both processors perform at most one non-q-step in each lap. This proves the claimed depth. □

Adding a suitable number of processors, the algorithm can be used to compute an addition chain γ in parallel for a set $\mathcal{E} = \{e^{(1)}, \ldots, e^{(t)}\} \subset \mathbb{N}_{\geq 1}$ in depth $\lceil \log_2 e \rceil$ where $e = \max \mathcal{E}$. While processor 0 successively performs all powers $2^0, \ldots, 2^{\lfloor \log_2 e \rfloor}$ as before, processor $p \in \{1, \ldots, t\}$ collects and sums up the intermediate results of processor 0 along the binary representation of $e^{(p)}$. This is illustrated in the left part of Figure 4.3.

COROLLARY 4.5. *Let $t \in \mathbb{N}_{\geq 1}$ and $\mathcal{E} = \{e^{(1)}, \ldots, e^{(t)}\} \subset \mathbb{N}_{\geq 1}$ be a set of integers with $e = \max \mathcal{E}$. Then there is an addition chain γ computing $\mathcal{E}$ in depth $\delta_{(1,1)}(\gamma) = \lceil \log_2 e \rceil$ using $t+1$ processors.*

If we compute all integers $\mathcal{E} = \{1, \ldots, q-1\}$ for $q \geq 3$ we can use the processors more sparingly than for an arbitrary set, see the right part of Figure 4.3. In lap $1 \leq i < \lceil \log_2(q-1) \rceil$, one can show by induction that exactly 2^{i-1} processors are enough to compute all nodes $2^{i-1}+1, \ldots, 2^i$ in lap i. The final lap $i = \lceil \log_2(q-1) \rceil$ adds only the nodes $2^{\lceil \log_2(q-1) \rceil - 1}+1, \ldots, q-1$ using $q-1-2^{\lceil \log_2(q-1) \rceil - 1}$ processors. Hence a total of $P = 2^{\lceil \log_2(q-1) \rceil - 1}$ processors is sufficient to compute all elements in $\mathcal{E} = \{1, \ldots, q-1\}$.

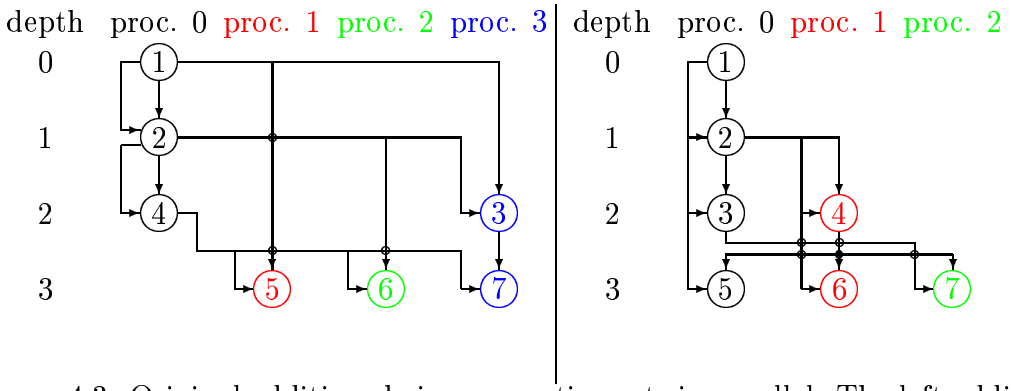


Figure 4.3: Original addition chains computing sets in parallel. The left addition chain illustrates Corollary 4.5 for the set $\mathcal{E} = \{5, 6, 7\}$. The right one is due to Corollary 4.6 for $\mathcal{E} = \{1, 2, \ldots, 7\}$.

COROLLARY 4.6. *Let $q \in \mathbb{N}_{\geq 3}$. An addition chain for the set $\mathcal{E} = \{1, \ldots, q-1\}$ can be computed in parallel in depth $\lceil \log_2(q-1) \rceil$ using at most $2^{\lceil \log_2(q-1) \rceil - 1} \leq q - 1$ processors.*

We now consider the general case $q \geq 2$ and $c_Q \geq c_A \geq 1$. For this situation the same algorithmic idea can be used to parallelize a q-addition chain with weight (c_Q, c_A). Then Algorithm 4.3 generalizes as follows: We first compute the set of digits of the q-ary representation of the exponent e in parallel using at most $q-1$ processors. After this *precomputation* we perform a *main stage* on q processors along the q-ary representation of e. All but one processor compute q-th powers to attach the right power of q to each digit e_i, $0 \leq i < \lambda_q(e)$. The remaining processor collects these intermediate results $e_i q^i$ and multiplies them.

ALGORITHM 4.7. Parallel exponentiation for $c_Q \geq c_A \geq 1$.

Input: A scalar $q \in \mathbb{N}_{\geq 2}$, and an exponent $e \in \mathbb{N}_{\geq 1}$ with q-ary representation $(e)_q = (e_{\lambda-1}, \ldots, e_0)$.

Output: A q-addition chain γ computing e.

1. Compute an addition chain γ for $\mathcal{E} = \{e_0, \ldots, e_{\lambda-1}\} \subseteq \{1, \ldots, q-1\}$.
2. For all processors $0 \leq p < P = q$ in parallel do 3–5
3. For $1 \leq i \leq \lceil \log_q(e) \rceil$ do 4–5
4. If $p < q-1$ and $\{i \leq j < \lambda_q(e) \colon e_j = p+1\} \neq \emptyset$ then add node $(p+1)q^i$ and rule $((p+1)q^{i-1}, -q)$ to γ.
5. If $p = q-1$ and $\sum_{0 \leq j < i} e_j q^j > e_{i-1} q^{i-1}$ then add node $\sum_{0 \leq j < i} e_j q^j$ and rule $(\sum_{0 \leq j < i-1} e_j q^j, e_{i-1} q^{i-1})$ to γ.
6. Return γ.

THEOREM 4.8. *Let $e \in \mathbb{N}_{\geq 1}$ be an exponent, $q \in \mathbb{N}_{\geq 2}$ be a scalar, and (c_Q, c_A) be a weight with $c_Q \geq c_A \geq 1$. Then there is a q-addition chain γ with weight*

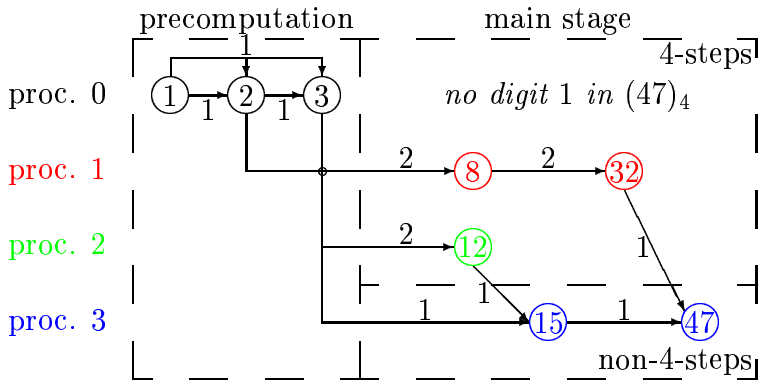


Figure 4.4: The graph of a 4-addition chain with weight $(2,1)$ computing 47 due to Algorithm 4.7.

(c_Q, c_A) computing e in depth

$$\delta_{q,(c_Q,c_A)}(e) \le \delta_{(c_Q,c_A)}(\gamma) \le c_A \cdot (\lceil \log_2(q-1) \rceil + 1) + c_Q \cdot (\lambda_q(e) - 1)\,.$$

This q-addition chain γ can be performed with q processors.

Proof. The proof of correctness is similar to the one given for Theorem 4.4. The invariant here is as follows: After lap $1 \le i < \lambda_q(e)$ the nodes $e_j q^j$ for $i \le j < \lambda_q(e)$ and $\sum_{0 \le j < i} e_j q^j$ if non-zero are in $\mathcal{S}(\gamma)$. The estimate on the depth follows by adding the depth $\lceil \log_2(q-1) \rceil$ to compute $\mathcal{E} \subseteq \{1, \ldots, q-1\}$ according to Corollary 4.6 and the depth of steps 3–5. But all processors perform either a q-step or a non-q-step or no step in each lap of the loop. By assumption we have $c_Q \ge c_A$ which gives $c_Q \cdot (\lambda_q(e) - 1)$. A final non-$q$-step may be added if $\lambda_q(e) = \lceil \log_q(e) \rceil$ and the claimed depth follows. □

4.1.3. Parallelizing the non-q-steps. We might also apply Algorithm 4.7 to the case $c_A > c_Q \ge 1$. Then the depth is no longer dominated by the q-steps in Algorithm 4.7 step 4. Now the non-q-steps—which are performed in step 5 by a single processor—become the bottleneck. To decrease the total depth, we therefore want to parallelize Algorithm 4.7 step 5. We suppose in this subsection that the elements of the set $\mathcal{E} = \{e_0, \ldots, e_{\lambda-1}\}$ are given at depth 0, i.e. the precomputation has already been done. While a single processor computes a single non-q-step, at most $\lceil \frac{c_A}{c_Q} \rceil$ new intermediate results $e_j q^j$ are ready for further use. But only one of these results is sufficient to keep a single processor working. The other $\lceil \frac{c_A}{c_Q} \rceil - 1$ intermediate results can be take from the stack by $\lceil \frac{c_A}{c_Q} \rceil - 1$ other processors. Since we concentrate only on the non-q-steps within the main stage now, we label the involved processors by $0 \le p < P' = \lceil \frac{c_A}{c_Q} \rceil$ here[21]. Our idea is thus to distribute the $\lambda_q(e)$ many intermediate results $e_j q^j$, $0 \le j < \lambda_q(e)$, equally

[21] Additionally, there are $q-1$ processors calculating the q-steps in parallel.

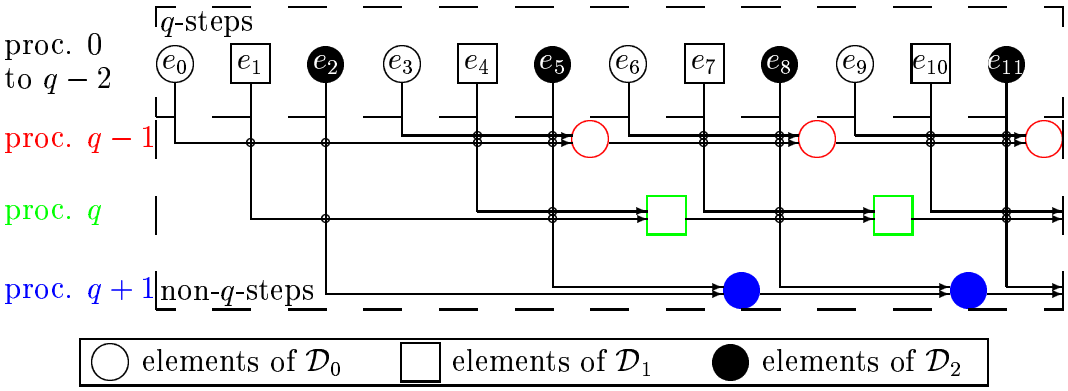


Figure 4.5: An illustration of the distribution of the non-q-steps for $c_A = 5$ and $c_Q = 2$. We can identify $\lceil \frac{5}{2} \rceil = 3$ sets of nodes such that each processor can perform a non-q-step before the next node is attached to this processor. All drawn edges have weight $c_A = 5$. We label only those nodes that are calculated by precomputation and q-th powers. Note e_j denotes $e_j q^j$.

to the P' processors. Since $c_Q \cdot \lceil \frac{c_A}{c_Q} \rceil \geq c_A$, each processor can complete a non-q-step before the next intermediate result $e_j q^j$ is attached to this processor. Our idea is illustrated in Figure 4.5. Summing up the results of these P' processors is a different task which we will discuss in the next paragraph. Formalizing this idea, we define a partition on the set of indices $\{0, 1, \ldots, \lambda_q(e) - 1\}$ by

$$\mathcal{D}_p = \{0 \leq j < \lambda_q(e) \colon j \equiv p \bmod P'\} \text{ for } 0 \leq p < P' = \left\lceil \frac{c_A}{c_Q} \right\rceil . \tag{4.9}$$

Then each processor computes the intermediate result

$$E_p = \sum_{j \in \mathcal{D}_p} e_j q^j \text{ for } 0 \leq p < P'$$

in depth at most $c_A + c_Q \cdot (\lambda_q(e) - 1)$. More precisely, we have the following:

Proposition 4.10. *Let e be an exponent, $e \geq 1$, q be a scalar greater than 1, and (c_Q, c_A) be a weight such that $c_A > c_Q \geq 1$. Let the set of digits $\mathcal{E} = \{e_0, \ldots, e_{\lambda-1}\}$ be given, and $\mathcal{D}_p$ for $0 \leq p < P' = \lceil \frac{c_A}{c_Q} \rceil$ be defined as in (4.9). Then the sum $E_p = \sum_{j \in \mathcal{D}_p} e_j q^j$ for an $0 \leq p < P'$ can be computed in parallel in depth at most*

$$c_A \cdot \Delta_p + c_Q \cdot \left(\lfloor (\lambda_q(e) - 1)/P' \rfloor \cdot P' + \Delta'_p \right)$$

where

$$\Delta_p = \begin{cases} 0 & \text{if } \lambda_q(e) - p \leq P' \\ 1 & \text{else,} \end{cases} \quad \text{and } \Delta'_p = \begin{cases} p+1 & \text{if } p \leq (\lambda_q(e) - 1) \operatorname{rem} P' \\ 0 & \text{else.} \end{cases}$$

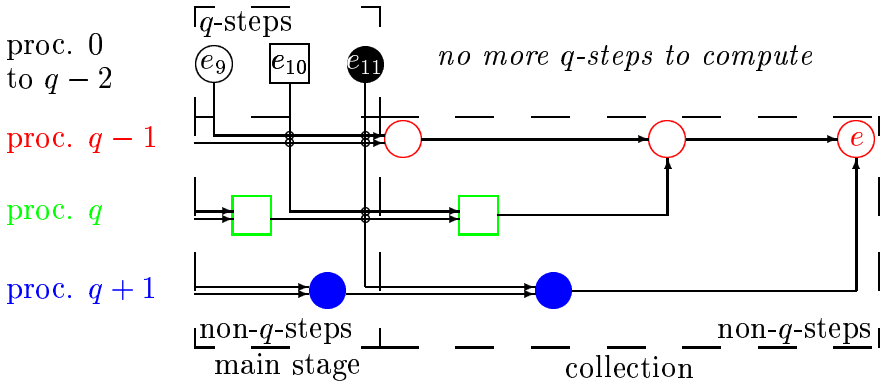


Figure 4.6: An illustration of the collection for $c_A = 5$ and $c_Q = 2$. Only edges with weight c_A are drawn. We have an addition along a tree with at most 3 leaves. This step is attached to the main branch-step, see Figure 4.5. It sums up the single results of the sets $\mathcal{D}_0$, $\mathcal{D}_1$, and $\mathcal{D}_2$.

PROOF. Since $c_Q \cdot \lceil \frac{c_A}{c_Q} \rceil \geq c_A$, we can compute a non-$q$-step while the next intermediate result $e_j q^j$ is calculated. Thus, the sum $E_p = \sum_{j \in \mathcal{D}_p} e_j q^j$ can be computed in depth at most $c_Q \cdot \max \mathcal{D}_p + c_A$. The final non-$q$-step is done if and only if $\#\mathcal{D}_p \geq 2$, i.e. $\lambda_q(e) \geq P' + p$ for $0 \leq p < P'$. It remains to determine $\max \mathcal{D}_p$ for each $0 \leq p < P'$. By construction we have $\lfloor \frac{\lambda_q(e)-1}{P'} \rfloor$ elements equally distributed between the sets $\mathcal{D}_0, \ldots, \mathcal{D}_{P'}$. The set $\mathcal{D}_p$, $0 \leq p < P'$, has one further element if and only if $(\lambda_q(e) - 1) - P' \cdot \lfloor \frac{\lambda_q(e)-1}{P'} \rfloor \geq p$ as claimed. □

4.1.4. Collecting the results. At most P' parts of the exponent are computed in parallel. These intermediate results $E_0, \ldots, E_{P'-1}$ have to be summed up in a subsequent stage. The most common method uses a binary tree with $\lfloor P'/2 \rfloor$ processors in parallel. Our task is a little bit more complicated, because not all leave nodes $E_p = \sum_{i \in \mathcal{D}_p} e_i q^i$ for $0 \leq p < P'$ may be available at depth 0. Proposition 4.10 states that if a summand is computed at depth d, its successor might be available not before depth $d + c_Q$. We discuss the scheme illustrated in Figure 4.6

PROPOSITION 4.11. *Let (c_Q, c_A) be a weight and P' be a positive integer such that $c_A \geq c_Q \cdot P' > 0$. Let $E_0, \ldots, E_{P'-1}$ be non-negative integers. If E_p is available in depth $c_Q \cdot p$ for $0 \leq p < P'$ then $\sum_{0 \leq p < P'} E_p$ can be computed in depth at most*

$$c_A \cdot A + c_Q \cdot \left(2^A - 1 - 2 \cdot (2^A - P')\right)$$

where $A = \lceil \log_2 P' \rceil$.

PROOF. We prove, by induction on the number of leaves $P' \in \mathbb{N}_{\geq 1}$, the following claim: If $E_0, \ldots, E_{P'-1}$ are available in depth $d_0 \leq \cdots \leq d_{P'-1}$, respectively,

with $|d_{P'-1} - d_0| \leq c_A$ then the sum $\sum_{0 \leq p < P'} E_p$ can be computed in depth at most $c_A \cdot A + d_{2^A - 1 - 2(2^A - P')}$ with $A = \lceil \log_2 P \rceil$.

The proposition follows then with $d_p = p \cdot c_Q$.

For $P' = 1$ there is no computation. Since $A = \lceil \log_2 1 \rceil = 0$ and $2^0 - 1 - 2(2^0 - 1) = 0$, the claim is true in this case. Thus suppose the induction hypothesis is valid for $1 \leq P < P'$.

If P' is even, we can perform a first computation $E'_p = E_{2p} + E_{2p+1}$ in depth $d_{p'} = d_{2p+1} + c_A$ for $0 \leq p < \frac{P'}{2}$. By the induction hypothesis, $\sum_{0 \leq p < \frac{P'}{2}} E'_p = \sum_{0 \leq p < P'} E_p$ can be computed in depth $c_A \cdot A' + d'_{2^{A'} - 1 - 2(2^{A'} - \frac{P'}{2})}$ with $A' = \lceil \log_2 \frac{P'}{2} \rceil$. But $d'_{2^{A'} - 1 - 2(2^{A'} - \frac{P'}{2})} = c_A + d_{2(2^{A'} - 1 - 2(2^{A'} - \frac{P'}{2}))+1} = c_A + d_{2^A - 1 - 2(2^A - P')}$ for $A = \lceil \log_2 P' \rceil = A' + 1$. The claim is true in this case.

Now, suppose P' is odd. We again compute a first addition step by $E'_{p+1} = E_{2p} + E_{2p+1}$ for $0 \leq p < \frac{P'-1}{2}$ in depth $d'_{p+1} = d_{2(p+1)-1} + c_A$. Furthermore, set $E'_0 = E_{P'-1}$ and $d'_0 = d'_1$. We have $d'_1 \geq d_{P'-1}$ since $|d_{P'-1} - d_0| \leq c_A$, by assumption; hence $d'_0 \leq \cdots \leq d'_{(P'+1)/2}$ and $|d'_{(P'+1)/2} - d'_0| \leq c_A$. The sum $\sum_{0 \leq p \leq (P'+1)/2} E'_p = \sum_{0 \leq p < P'} E_p$ can be computed in depth $A' \cdot c_A + d'_{2^{A'} - 1 - 2(2^{A'} - (P'+1)/2)}$ with $A' = \lceil \log_2 \frac{P'+1}{2} \rceil$, by the induction hypothesis . Since $P' \geq 3$ is odd, we have $A' = -1 + \lceil \log_2(P'+1) \rceil = \lceil \log_2 P' \rceil - 1 = A - 1$, and $d'_{2^{A'} - 1 - 2(2^{A'} - (P'+1)/2)} = c_A + d_{2(2^{A'} - 1 - 2(2^{A'} - (P'+1)/2)) - 1} = c_A + d_{2^A - 1 - 2(2^A - P')}$ as claimed. □

We summarize the above and describe the complete algorithm giving a weighted q-addition chain for an exponent e in the case $c_A > c_Q \geq 1$.

ALGORITHM 4.12. Parallel exponentiation for $c_A > c_Q \geq 1$.

Input: A scalar $q \in \mathbb{N}_{\geq 2}$, an exponent $e \in \mathbb{N}_{\geq 1}$ with q-ary representation $(e)_q = (e_{\lambda-1}, \ldots, e_0)$, and a weight (c_Q, c_A) with $c_A > c_Q \geq 1$.

Output: A q-addition chain γ computing e.

1. Compute an addition chain γ for $\mathcal{E} = \{e_0, \ldots, e_{\lambda-1}\} \subseteq \{1, \ldots, q-1\}$.
2. Set $P_0 = q - 1$ and $P_1 = \min\{\lceil \frac{c_A}{c_Q} \rceil, \lambda_q(e)\}$.
3. For all processors $0 \leq p < P = P_0 + P_1$ in parallel do 4–16
4. For $1 \leq i \leq \lceil \log_2 e \rceil$ do 5–6
5. If $0 \leq p < P_0$ and $\{i \leq j < \lambda_q(e) \colon e_j = p + 1\} \neq \emptyset$ then add node $(p+1)q^i$ and rule $((p+1)q^{i-1}, -q)$ to γ.
6. If $P_0 \leq p < P_0 + P_1$ and $i - 1 \in \mathcal{D}_{p-P_0}$ and $\sum_{0 \leq j < i} e_j q^j > e_{i-1} q^{i-1}$ then add node $\sum_{0 \leq j < i, j \in \mathcal{D}_{p-P_0}} e_j q^j$ and rule $(\sum_{0 \leq j < i-1, j \in \mathcal{D}_{p-P_0}} e_j q^j, e_{i-1} q^{i-1})$ to γ.
7. If $p \geq P_0$ then
8. Set $P' = P_1$ and $E_p = \sum_{j \in \mathcal{D}_p} e_j q^j$ for $0 \leq p < P'$.
9. For $1 \leq i \leq \lceil \log_2 P_1 \rceil$ do 10–16
10. If P' is even then
11. Set $P' \leftarrow \frac{P'}{2}$.

12. If $p < P'$ then add node $E_{2p} + E_{2p+1}$ and rule (E_{2p}, E_{2p+1}) to γ. Set $E_p \leftarrow E_{2p} + E_{2p+1}$.
13. Else
14. Set $P' \leftarrow \frac{P'+1}{2}$.
15. If $p < P' - 1$ then add node $E_{2p} + E_{2p+1}$ and rule (E_{2p}, E_{2p+1}) to γ. Set $E_{p+1} \leftarrow E_{2p} + E_{2p+1}$.
16. If $p = P' - 1$ then set $E_0 \leftarrow E_{2P'-1}$.
17. Return γ.

THEOREM 4.13. *Let e be an exponent, $e \geq 1$, q be a scalar greater than 1, and (c_Q, c_A) be a weight with $c_A > c_Q \geq 1$. Then there exists a q-addition chain γ with weight (c_Q, c_A) computing e in depth at most*

$$\begin{aligned}\delta_{q,(c_Q,c_A)}(e) \;\leq\; & \delta_{(c_Q,c_A)}(\gamma) \leq c_Q \cdot (\lambda_q(e) - 1) \\ & + c_A \cdot \left(\lceil \log_2(q-1) \rceil + 1 + \left\lceil \log_2 \min \left\{ \left\lceil \frac{c_A}{c_Q} \right\rceil, \lambda_q(e) \right\} \right\rceil \right).\end{aligned}$$

The q-addition chain can be performed on at most $q - 1 + \min\{\lceil \frac{c_A}{c_Q} \rceil, \lambda_q(e)\}$ processors.

PROOF. Algorithm 4.12 is just Algorithm 4.7 with additional steps 9–16. These additional steps implement Proposition 4.11. Therefore correctness follows.

The estimate on the depth can be derived as follows:

- Step 1 can be computed in depth $\lceil \log_2(q-1) \rceil$ on at most $q-1$ processors according to Corollary 4.6.

- Steps 4–6 can be performed in depth at most $c_Q \cdot (\lambda_q(e) - 1) + c_A$ by Proposition 4.10: $e_{\lambda-1} q^{\lambda-1}$ is computed in depth $c_Q \cdot (\lambda_q(e) - 1)$, and there might be a subsequent non-q-step. The number of processors is $P' = \min\{\lceil \frac{c_A}{c_Q} \rceil, \lambda_q(e)\}$. If there are less than $\lceil \frac{c_A}{c_Q} \rceil$ digits in $(e)_q$, we can restrict to $P' = \lambda_q(e)$ processors and distribute each digit $e_p q^p$ to processor p for $0 \leq p < P'$.

- Proposition 4.11 yields that $E_0, \ldots, E_{P'-1}$ can be computed in depth $c_A \cdot \lceil \log_2 P' \rceil + c_Q \cdot (2^{\lceil \log_2 P' \rceil} - 1 - 2(2^{\lceil \log_2 P' \rceil} - P'))$ with at most P' processors.

The computation of steps 9–16 can be started on the same processors that execute steps 4–6 after the first intermediate result E_p is available. This is given in depth at most $c_A + c_Q \cdot (\lambda_q(e) - 1 - (P' - 1))$ since the last processor completes its calculation in depth at most $c_Q \cdot (\lambda_q(e) - 1)$. Thus, we have the following upper

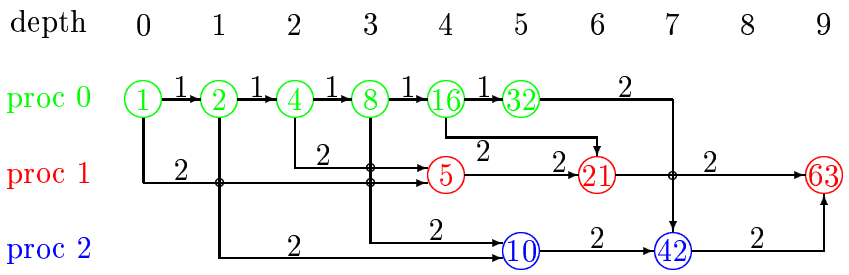


Figure 4.7: Illustration of Algorithm 4.12 for $q = 2$, weight $(1, 2)$ and exponent $(63)_2 = (111111)$.

bound on the depth:

$$\begin{aligned} & c_A \cdot \lceil \log_2(q-1) \rceil + c_A + c_Q \cdot (\lambda_q(e) - 1) - c_Q \cdot (P' - 1) \\ & + c_A \cdot (\lceil \log_2 P' \rceil + c_Q \cdot (2^{\lceil \log_2 P' \rceil} - 1 - 2 \cdot (2^{\lceil \log_2 P' \rceil} - P')) \\ \leq\ & c_A \cdot (\lceil \log_2(q-1) \rceil + 1 + \lceil \log_2 P' \rceil) + c_Q \cdot (\lambda_q(e) - 1) \\ & + c_Q \cdot (-2^{\lceil \log_2 P' \rceil} - 1 + 2P' - P' + 1) \\ \leq\ & c_A \cdot (\lceil \log_2(q-1) \rceil + 1 + \lceil \log_2 P' \rceil) + c_Q \cdot (\lambda_q(e) - 1). \end{aligned}$$ □

4.1.5. Free scalar. The only case not covered neither by Theorem 4.8 nor by Theorem 4.13 is the case $c_Q = 0$. Let $\omega_q(e)$ be the sum of digits of the q-ary representation of e, i.e. $\omega_q(e) = \sum_{0 \leq i < \lambda} e_i$, and let $0 \leq i(0) \leq \cdots \leq i(\omega_q(e)-1) \leq (q-1) \cdot \lambda_q(e)$ be a sequence of integers such that $\sum_{0 \leq p < \omega_q(e)} q^{i(p)} = e$. Then $q^{i(p)}$ is available in depth 0 for all $0 \leq p < \omega_q(e)$ and $\sum_{0 \leq p < \omega_q(e)} q^{i(p)} = e$ can be computed in depth at most $\lceil \log_2(\omega_q(e)) \rceil$ by Proposition 4.11. Thus, in the case $c_Q = 0$, we can omit the precomputation of the digits $e_0, \ldots, e_{\lambda-1}$ and the separation into branches. The resulting algorithm was described by von zur Gathen (1991).

ALGORITHM 4.14. Parallel exponentiation with free scalar.
Input: An exponent $e \in \mathbb{N}_{\geq 1}$ and a scalar $q \in \mathbb{N}_{\geq 2}$.
Output: A q-addition chain γ with weight $(0, 1)$ computing e.

1. Set γ the empty addition chain with $\mathcal{S}(\gamma) = \{1\}$.
2. Set $P = \omega_q(e)$ and $\lambda = \lambda_q(e)$ and let $0 \leq i(0) \leq i(1) \leq \cdots \leq i(P-1) \leq \lambda$ be a sequence of integers such that $\sum_{0 \leq p < P} q^{i(p)} = e$.
3. For all processors $0 \leq p < P$ in parallel do 4–6
4. Set $E_p = q^{i(p)}$.
5. For $1 \leq i \leq \lceil \log_2 \omega_q(e) \rceil$ do
6. If 2^i divides p and $p + 2^{i-1} < P$ then set $E_p \leftarrow E_p + E_{p+2^{i-1}}$ and add node E_p and rule $(E_p - E_{p+2^{i-1}}, E_{p+2^{i-1}})$ to γ.
7. Return γ.

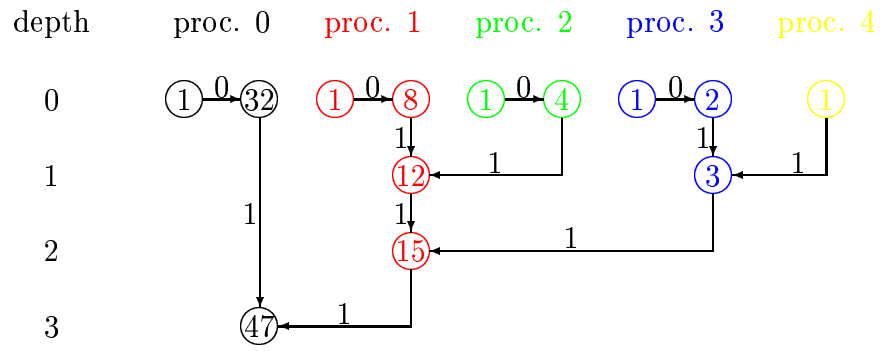


Figure 4.8: The graph of a 2-addition chain with free doublings computing 47 due to Algorithm 4.14.

FACT 4.15 (von zur Gathen 1991, Theorem 4.3). *Let e be an exponent, $e \geq 1$, and q be a scalar greater than 1. Then there is a q-addition chain γ of weight $(0,1)$ computing e such that*

$$\delta_{q,(0,1)}(e) \leq \delta_{(0,1)}(\gamma) = \lceil \log_2 \omega_q(e) \rceil$$

using $\lfloor \frac{\omega_q(e)}{2} \rfloor$ processors.

It remains to derive Corollary 4.1 from the results above.

PROOF (of Corollary 4.1). For exponentiation in $\mathbb{F}_{q^n}$, we suppose $0 \leq e < q^n$ by Fermat's Little Theorem 2.3, i.e. $\lambda_q(e) = \lfloor \log_q(e) \rfloor + 1 \leq \lfloor \log_q(q^n - 1) \rfloor + 1 \leq n$. Furthermore, the sum of digits $\omega_q(e) = \sum_{0 \leq i < \lambda} e_i \leq \sum_{0 \leq i < \lambda} q - 1 = (q-1)\lambda_q(e)$ is bounded by $(q-1)n$. The case $c_Q \geq c_A \geq 1$ (part (i)) follows with Theorem 4.8, and Theorem 4.13 covers the case $c_A > c_Q \geq 1$ (part (ii)). Part (iii) is Corollary 4.24 with respect to $\mathbb{F}_{q^n}$. □

4.2. A lower bound for parallel exponentiation. We again consider the question: How fast can we evaluate a power in parallel? Now we prove a lower bound for our model of weighted q-addition chains extending the presentation in Nöcker (2000). For this bound, the following number will be useful.

NOTATION 4.16. *Let q be a scalar greater than 1, and (c_Q, c_A) be a weight. For an exponent $e \in \mathbb{N}_{\geq 1}$ with q-ary representation $(e)_q = (e_{\lambda-1}, \ldots, e_0)$ we set*

$$[e]_{q,c} = \sum_{0 \leq i < \lambda} e_i (2^c)^i$$

and call $[e]_{q,c}$ the scaling of $(e)_q$ with respect to $c = \frac{c_Q}{c_A}$.

Informally spoken, the scaling $[e]_{q,c}$ is generated by substituting q by 2^c in $e = \sum_{0\le i<\lambda} e_i q^i$. It will turn out below that this scaling is a measure of difficulty on the parallel computation of a q-addition chain with weight (c_Q, c_A) for e. It is our key to prove a new lower bound for parallel exponentiation in the model of weighted q-addition chains.

RESULT 4.17. *Let e be an exponent, $e \ge 1$, q be a scalar greater than 1, and (c_Q, c_A) be a weight with $0 \le c = \frac{c_Q}{c_A} \le \log_2 q$. Set $Q = \lambda_q(e) - 1$ and $A = \lceil \log_2 [e]_{q,c} - Q \cdot c \rceil$. Any q-addition chain γ with weight (c_Q, c_A) computing e has depth at least*

$$\delta_{q,(c_Q,c_A)}(e) \begin{cases} > c_Q \cdot Q + c_A \cdot (A-1) & \text{if } A \ge 1, \text{ and} \\ \ge c_Q \cdot Q & \text{if } A = 0. \end{cases}$$

Some illustrations. Before we will prove this new lower bound on *parallel weighted q-addition chains*, we illustrate the formulas. We start with the scaling of $(e)_q$ for special values of c.

EXAMPLE 4.18. Let q be an integer greater than 1.

(i) For $c_Q = 0$, i.e. $c = 0$, the scaling of $(e)_q$ is given by

$$[e]_{q,0} = \sum_{0\le i<\lambda} e_i (2^0)^i = \sum_{0\le i<\lambda} e_i = \omega_q(e).$$

Thus, if $c = 0$ then the scaling of $(e)_q$ is just the sum of digits $\omega_q(e)$. We have $\omega_q(e) \le (q-1) \cdot \lambda_q(e)$, i.e. the scaling is logarithmically bounded by e.

(ii) For $c_Q = c_A \cdot \log_2 q$, i.e. $c = \log_2 q$, the scaling of $(e)_q$ is just e, since

$$[e]_{q,\log_2 q} = \sum_{0\le i<\lambda} e_i (2^{\log_2 q})^i = \sum_{0\le i<\lambda} e_i q^i = e. \qquad \Diamond$$

For $0 < c < \log_2 q$ we have not found a similar easy description of $[e]_{q,c}$. But we can bound it with regard to $(2^c)^{\lambda-1}$ since

$$\begin{aligned} & 1 \cdot (2^c)^{\lambda-1} \le e_{\lambda-1} \cdot (2^c)^{\lambda-1} \le [e]_{q,c} = \sum_{0\le i<\lambda} e_i (2^c)^{\lambda-1} \\ \le & \sum_{0\le i<\lambda} \max\{e_j \colon 0 \le j < \lambda\} \cdot (2^c)^i = \max\{e_j \colon 0 \le j < \lambda\} \cdot \frac{(2^c)^\lambda - 1}{2^c - 1} \\ \le & (q-1) \cdot \left(\frac{2^c}{2^c - 1} \cdot (2^c)^{\lambda-1} - \frac{1}{2^c - 1} \right) < \frac{(q-1) \cdot 2^c}{2^c - 1} \cdot (2^c)^{\lambda-1}. \end{aligned}$$

The number $(2^c)^{\lambda-1}$ is the scaling of the integer $e = q^{\lambda-1}$. It can be computed with $\lambda - 1$ many q-steps in depth $c_Q \cdot (\lambda - 1)$. In Section 4.1, all exponents with

$\lambda_q(e)$ digits used $\lambda_q(e)-1$ many q-steps and "some additional non-q-steps". Due to this observation, we distinguish in Result 4.17 between $\lambda_q(e)-1$ many q-steps on the one hand, and non-q-steps on the other hand. Since

$$\begin{aligned} A &= \lceil \log_2 [e]_{q,c} - c \cdot Q \rceil = \lceil \log_2 \sum_{0 \le i < \lambda} e_i (2^c)^i - \log_2 (2^c)^{\lambda-1} \rceil \\ &= \lceil \log_2 \sum_{0 \le i < \lambda} e_{\lambda-1-i} (2^c)^{-i} \rceil \end{aligned}$$

and $e_{\lambda-1} \cdot (2^c)^0 = e_{\lambda-1} \ge 1$, we have $A = 0$ if and only if e is a power of q. Thus, the order of A is related to the non-zero digits of e.

EXAMPLE 4.18 CONTINUED. For the two special cases discussed above the result reads as follows:

(i) For $c = 0$, we have

$$A = \lceil \log_2 [e]_{q,0} - 0 \cdot Q \rceil = \lceil \log_2 \omega_q(e) \rceil.$$

Moreover, the atomic unit is one non-q-step with cost c_A. Thus, for $A > 0$ we can substitute $\delta_{q,(0,c_A)}(e) > c_A \cdot (A-1)$ by $\delta_{q,(0,c_A)} \ge c_A \cdot A = c_A \cdot \lceil \log_2 \omega_q(e) \rceil$.

(ii) For $c = \log_2 q$, the minimal number of additional non-q-steps is at most

$$\begin{aligned} A &= \lceil \log_2 [e]_{q,\log_2 q} - (\lambda_q(e) - 1) \cdot \log_2 q \rceil \\ &= \left\lceil \log_2 e - \log_2 q \cdot \left(\left\lfloor \frac{\log_2 e}{\log_2 q} \right\rfloor + 1 - 1 \right) \right\rceil \\ &\le \left\lceil \log_2 e - \log_2 q \cdot \left(\frac{\log_2 e}{\log_2 q} - 1 \right) \right\rceil = \lceil \log_2 q \rceil. \end{aligned}$$

For $q = 2$ we have $A + Q = \lceil \log_2 e \rceil$ and the bound can be written as $\delta_{2,(1,1)}(e) \ge \lceil \log_2 e \rceil$. ◊

The two special cases discussed in Example 4.18 can already be found in the literature. We will show below that our general result meets both the lower bound for original addition chains given by Kung (1976), Theorem 4.1, and for addition chains with free scalar by von zur Gathen (1991), Theorem 4.3.[22] For $0 < c < \log_2 q$, we have not found a closed form for A depending on q, c and e. But as above, we can bound the maximal value for A by $\lceil \log_2 (q-1) \rceil + \lceil \log_2 \frac{2^c}{2^c-1} \rceil$.

[22] Both von zur Gathen (1991) and Kung (1976) proved their bounds in models which are more general than addition chains. In Theorem 4.1, von zur Gathen (1991) gave another version for arithmetic circuits allowing $+$, $-$, $\cdot$ and free q-th powers. Kung (1976) formulated his Theorem 4.1 for rational expressions over a field $\mathbb{F}$ considering only addition, multiplication and division of two elements in $\mathbb{F}(x)$.

REMARK 4.19. *The lower bound given in Result 4.17 is not always sharp for* $0 < c < \log_2 q$. *For* $q = 2$ *and the weight* $(1, 2)$, *i.e.* $c = \frac{1}{2}$, *the lower bound on the exponent* $(63)_2 = (111111)$ *is*

$$\delta_{2,(1,2)}(e) > 1 \cdot (6 - 1) + 2 \cdot (2 - 1) = 7,$$

since $Q = \lambda_2(63) - 1 = 6 - 1 = 5$ *and* $A = \lceil \log_2(1 + \sqrt{2}^{-1} + 2^{-1} + (2\sqrt{2})^{-1} + 4^{-1} + (4\sqrt{2})^{-1}) \rceil = \lceil \log_2(14 + 7\sqrt{2}) - \log_2 8 \rceil = 5 - 3 = 2$.

Algorithm 4.12 computes a parallel 2-addition chain with weight $(1, 2)$ *in depth* 9, *see Figure 4.7. This depth is indeed minimal as we have shown by computations. The table below lists all exponents up to* 64 *and the depth of a shortest parallel addition chain for these exponents.*

depth	*computed exponents*
0	1
1	2
2	4
3	3, 8
4	5, 6, 16
5	7, 9, 10, 11, 12, 32
6	13, 14, 17, 18, 19, 20, 21, 22, 24, 64
7	15, 23, 25, 26, 27, 28, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 48
8	29, 30, 31, 45, 46, 49, 50, 51, 52, 53, 54, 56
9	47, 55, 57, 58, 59, 60, 61, 62, **63**

4.2.1. The basic idea. Original addition chains are a good way to introduce our idea. In this case, we can restrict to non-q-steps and may assume, without loss of generality, that $c_A = 1$. Let γ be an addition chain with $\mathcal{S}(\gamma) = \{1, 2, \ldots, e\}$ for an $e \in \mathbb{N}_{\geq 1}$. Then we can sort all exponents $e' \in \mathcal{S}(\gamma)$ by their minimal depth assuming suitable rules as illustrated in Figure 4.9. This motivates the following fact which is a growth argument on the exponents.

FACT 4.20 (Kung 1976). *Performing only non-q-steps with unit cost* $c_A = 1$, *all exponents* $e \in \mathbb{N}_{\geq 1}$ *that can be computed in depth* $d \in \mathbb{N}_{\geq 0}$ *are less or equal* 2^d.

PROOF. The proof is by induction on the depth $d \in \mathbb{N}_{\geq 0}$. For $d = 0$ the only possible exponent is $a_0 = 1$. Hence, in this case the claim is true, and we suppose that the assumption is true for $0 \leq d' < d$. Let a_i be computed in depth at most d. If a_i is already computed in depth d' there is nothing to prove. Therefore, we can assume a_i to be computed in depth exactly d. Then there are $a_{j(i)}, a_{k(i)} \in \mathbb{N}_{\geq 0}$ with $a_i = a_{j(i)} + a_{k(i)}$. Both $a_{j(i)}$ and $a_{k(i)}$ are computed in depth at most $d - 1$. By the induction hypothesis, we have $a_{j(i)}, a_{k(i)} \leq 2^{d-1}$ and therefore $a_i = a_{j(i)} + a_{k(i)} \leq 2^{d-1} + 2^{d-1} = 2^d$. This completes the proof. □

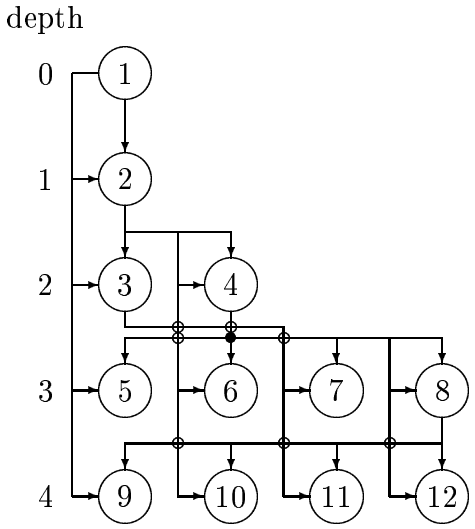


Figure 4.9: A linear addition chain computing $\{1, \ldots, 12\}$ in parallel. All nodes have minimal depth.

The remark contains a lower bound on the depth of any original addition chain γ that computes an exponent $e \in \mathbb{N}_{\geq 1}$ in parallel. There is a unique $d \in \mathbb{N}_{\geq 0}$ such that $2^{d-1} < e \leq 2^d$, i.e. $d - 1 < \log_2 e \leq d$ or just $d = \lceil \log_2 e \rceil$. This conclusion leads to Theorem 4.1 in Kung (1976) for original addition chains.

COROLLARY 4.21 (Kung 1976). *Let e be an exponent, $e \geq 1$. Any addition chain γ computing e has depth at least*

$$\delta_{2,(1,1)}(e) \geq \lceil \log_2 e \rceil.$$

This bound meets the upper bound in Theorem 4.4, therefore it is optimal.

4.2.2. An extension to weighted addition chains. We want to investigate our model of weighted q-addition chains. We now have to consider two different types of steps: non-q-steps with cost $c_A \in \mathbb{N}_{\geq 1}$ and q-steps with cost $c_Q \in \mathbb{N}_{\geq 0}$. Since a q-step can be substituted by non-q-steps, we can suppose without loss of generality that $0 \leq c_Q \leq c_A \cdot \log_2 q$ by Algorithm 4.3. Observe that

$$\{c_Q \cdot Q + c_A \cdot A \colon Q, A \in \mathbb{N}_{\geq 0}\} \subseteq \{\gcd(c_Q, c_A) \cdot d \colon d \in \mathbb{N}_{\geq 0}\}.$$

Thus any algorithm performing only q-steps and non-q-steps has depth a multiple of $\gcd(c_Q, c_A)$. Substituting the original weight by $(\frac{c_Q}{\gcd(c_Q,c_A)}, \frac{c_A}{\gcd(c_Q,c_A)})$ preserves the relation $c = \frac{c_Q}{c_A}$. The original depth is then the actual depth times $\gcd(c_Q, c_A)$. Thus without loss of generality, we confine ourselves to the case $\gcd(c_Q, c_A) = 1$. Each level of depth is then labeled by $d = c_Q \cdot Q + c_A \cdot A$ for $d \in \mathbb{N}_{\geq 0}$. This is the depth in which at most $A \in \mathbb{N}_{\geq 0}$ non-q-steps and $Q \in \mathbb{N}_{\geq 0}$ many q-steps can be computed. Therefore Fact 4.20 extends to the following statement.

THEOREM 4.22. *Let q be a scalar greater than 1, and (c_Q, c_A) be a weight with $\gcd(c_Q, c_A) = 1$ and $0 \leq c = \frac{c_Q}{c_A} \leq \log_2 q$. Only performing non-$q$-steps with cost c_A and q-steps with cost c_Q, all exponents $e \in \mathbb{N}_{\geq 1}$ that can be computed in depth $d \in \mathbb{N}_{\geq 0}$ satisfy*

$$[e]_{q,c} = \sum_{0 \leq i < \lambda} e_i (2^c)^i \leq 2^{d/c_A}$$

where $e_0, \ldots, e_{\lambda-1}$ are the digits of the q-ary representation of e.

For $c_Q = 0$, we have $c = \frac{0}{c_A} = 0$ and $[e]_{q,0} = \sum_{0 \leq i < \lambda} e_i (2^0)^i = \sum_{0 \leq i < \lambda} e_i$. This is the sum of digits $\omega_q(e)$ of $(e)_q$. Since $\gcd(0, c_A) = c_A$, we can rewrite Theorem 4.22 for the special case of free scalars.

PROPOSITION 4.23 (von zur Gathen 1991). *If the only used operations are non-q-steps with cost $c_A = 1$ and free q-steps then all exponents $e \in \mathbb{N}_{\geq 1}$ that can be computed in depth $d \in \mathbb{N}_{\geq 0}$, satisfy $\omega_q(e) \leq 2^d$.*

PROOF. Note that for $e, f \in \mathbb{N}_{\geq 1}$ we have

$$\begin{aligned} \omega_q(q \cdot e) &= \omega_q(e) \text{ and} \\ \omega_q(e + f) &\leq \omega_q(e) + \omega_q(f) \leq 2 \cdot \max\{\omega_q(e), \omega_q(f)\}. \end{aligned}$$

We prove the claim by induction on $d \in \mathbb{N}_{\geq 0}$. All exponents that can be computed in depth $d = 0$ are computed without non-q-steps. Hence, only powers of q are possible which have got Hamming weight $1 \leq 2^0$. Now, we assume that the induction hypothesis holds for depth $0 \leq d' < d$ and let a_i be computed in depth at most d. If a_i is already calculated at a lower level than d, there is nothing to prove. Thus, we suppose that a_i is indeed computed at level d. If a_i is divisible by q, we focus on $\frac{a_i}{q^m}$ with $m = \max\{m' \in \mathbb{N}_{\geq 1} : q^{m'} \text{ divides } a_i\}$ instead of a_i. The m subsequent q-steps can be done without cost. Therefore, we can assume the last step to be a non-q-step. For $d \geq 1$, there are $a_{j(i)}, a_{k(i)} \in \mathbb{N}_{\geq 1}$ such that $a_i = a_{j(i)} + a_{k(i)}$. Both $a_{j(i)}$ and $a_{k(i)}$ are calculated in depth at most $d - c_A = d - 1$. By the induction hypothesis, we have $\omega_q(a_{j(i)}), \omega_q(a_{k(i)}) \leq 2^{d-1}$, and therefore, $\omega_q(a_i) \leq \omega_q(a_{j(i)}) + \omega_q(a_{k(i)}) \leq 2^{d-1} + 2^{d-1} = 2^d$. □

Again, Proposition 4.23 contains a lower bound which was originally stated by von zur Gathen (1991), Theorem 4.3.

COROLLARY 4.24 (von zur Gathen 1991). *Let e be an exponent, $e \geq 1$, and q be a scalar greater than 1. Any q-addition chain γ with weight $(0, 1)$ that computes e has depth at least*

$$\delta_{q,(0,1)}(e) \geq \lceil \log_2 \omega_q(e) \rceil.$$

This is also the upper bound by Fact 4.15. Therefore, it is optimal. Now, we return to the case $c_Q > 0$ and give a proof of Theorem 4.22.

PROOF (Theorem 4.22). For $c_Q = 0$, we refer to Proposition 4.23. We prove the remaining case $c = \frac{c_Q}{c_A} > 0$ by induction on $d \in \mathbb{N}_{\geq 0}$. For $d = 0$, the only computable exponent is $a_0 = 1$. But $[1]_{q,c} = 1 \cdot (2^c)^0 = 1 \leq 2^0$, and the claim is true for $d = 0$. Now, we suppose that the claim is also true for $0 \leq d' < d$. Let e be an exponent computed at depth at most d. If e is calculated in depth less than d then there is nothing to prove. Thus, let e be computed in depth exactly d. Then the final step is either a q-step or a non-q-step.

- Case: *e is computed by a final q-step*
  Then, we have a rule $(\frac{e}{q}, -q)$, and $f = \frac{e}{q}$ has been calculated in depth at most $d - c_Q$. By the induction hypothesis, we know that $[f]_{q,c} \leq 2^{(d-c_Q)/c_A}$. We observe that $(e)_q = (f_{\lambda-1}, \ldots, f_0, 0)$ which gives

$$\begin{aligned}[e]_{q,c} &= \sum_{0 \leq i < \lambda_q(e)} e_i (2^c)^i = 0 + \sum_{1 \leq i < \lambda_q(e)} f_{i-1} (2^c)^i = 2^c \cdot [f]_{q,c} \\ &\leq 2^c \cdot 2^{d/c_A - c_Q/c_A} = 2^c \cdot 2^{d/c_A - c} = 2^{d/c_A}\end{aligned}$$

  and proves the claim.

- Case: *e is computed by a final non-q-step*
  There are $f, g \in \mathbb{N}_{\geq 1}$ such that $e = f + g$ and both summands are computed in depth at most $d - c_A$. Without loss of generality we suppose that $\lambda = \lambda_q(f) \geq \lambda_q(g)$. We set $g_j = 0$ for $\lambda_q(g) \leq j < \lambda_q(f)$ in the q-ary representation of g, and define the carries $u_0 = 0$ and $u_{i+1} = \lfloor (f_i + g_i + u_i)/q \rfloor \in \{0, 1\}$ for $0 \leq i < \lambda$. Then $e_i = f_i + g_i + u_i - q u_{i+1}$ for $0 \leq i < \lambda$ and $e_\lambda = c_\lambda$. This gives

$$\begin{aligned}[e]_{q,c} &= \sum_{0 \leq i \leq \lambda} e_i (2^c)^i = u_\lambda (2^c)^\lambda + \sum_{0 \leq i < \lambda} (f_i + g_i + u_i - q u_{i+1}) (2^c)^i \\ &= \sum_{0 \leq i < \lambda} f_i (2^c)^i + \sum_{0 \leq i < \lambda} g_i (2^c)^i - \sum_{1 \leq i \leq \lambda} u_i (q - 2^c)(2^c)^{i-1} \\ &= [f]_{q,c} + [g]_{q,c} - \sum_{1 \leq i \leq \lambda} u_i (q - 2^c)(2^c)^{i-1} \\ &\leq 2 \cdot 2^{(d-c_A)/c_A} = 2^{d/c_A},\end{aligned}$$

  by the induction hypothesis, since $c = \frac{c_Q}{c_A} \leq \log_2 q$, i.e. $q \geq 2^c$. This completes the proof for the second case. □

As for the two special cases discussed above, a lower bound on weighted q-addition chains is covered by Theorem 4.22. This bound is already claimed in Result 4.17. We are now ready to give the proof.

PROOF (of Result 4.17). For an exponent $e \in \mathbb{N}_{\geq 1}$ with q-ary representation $(e)_q = (e_{\lambda-1}, \ldots, e_0)$, we set

$$\begin{aligned} Q &= \lambda - 1 \text{ and} \\ A &= \lceil \log_2 [e]_{q,c} - Q \cdot c \rceil \\ &= \left\lceil \log_2 \sum_{0 \leq j < \lambda} e_j (2^c)^j - \log_2 (2^c)^Q \right\rceil = \left\lceil \log_2 \sum_{0 \leq j < \lambda} e_j (2^c)^{j-Q} \right\rceil, \end{aligned}$$

where $c = \frac{c_Q}{c_A}$. Let $d = c_Q \cdot Q + c_A \cdot A$. Obviously, we have

$$\begin{aligned} 2^{d/c_A} &= 2^{c_Q Q/c_A} \cdot 2^{c_A A/c_A} = (2^c)^Q \cdot 2^A = (2^c)^Q \cdot 2^{\lceil \log_2 \sum_{0 \leq j < \lambda} e_j (2^c)^{j-Q} \rceil} \\ &\geq (2^c)^Q \cdot 2^{\log_2 \sum_{0 \leq j < \lambda} e_j (2^c)^{j-Q}} = (2^c)^Q \cdot \sum_{0 \leq j < \lambda} e_j (2^c)^{j-Q} \\ &= \sum_{0 \leq j < \lambda} e_j (2^c)^j = [e]_{q,c}. \end{aligned}$$

If $A \geq 1$, then

$$\begin{aligned} 2^{(d-c_A)/c_A} &= 2^{c_Q Q/c_A} \cdot 2^{c_A (A-1)/c_A} = (2^c)^Q \cdot 2^{\lceil \log_2 \sum_{0 \leq j < \lambda} e_j (2^c)^{j-Q} \rceil - 1} \\ &< (2^c)^Q \cdot 2^{\log_2 \sum_{0 \leq j < \lambda} e_j (2^c)^{j-Q}} = \sum_{0 \leq j < \lambda} e_j (2^c)^j = [e]_{q,c}, \end{aligned}$$

and thus, any q-addition chain with weight (c_Q, c_A) computing e has depth more than $d - c_A = c_Q \cdot Q + c_A \cdot (A-1)$ if $A \geq 1$. If $A = 0$ and $c_Q Q \geq 0$ then we have

$$\begin{aligned} 2^{d/c_A} &= 2^{c_Q Q/c_A} \cdot 2^0 = (2^c)^Q = (2^c)^{\lambda-1} \\ &\leq e_{\lambda-1} (2^c)^{\lambda-1} \leq \sum_{0 \leq j < \lambda} e_j (2^c)^j = [e]_{q,c}. \end{aligned}$$

Thus any q-addition chain with weight $(c_Q, c_A) \in \mathbb{N}_{\geq 1} \times \mathbb{N}_{\geq 1}$ computing e has depth at least $d = c_Q \cdot Q$ if $A = 0$. □

Any q-addition chain has depth at least this lower bound plus 1 if $A \geq 1$. Therefore, for $c_A = c_Q = 1$, we have to compute at least

$$\begin{aligned} d &= 1 \cdot Q + 1 \cdot A = \lambda - 1 + \lceil \log_2 [e]_{2,1} - Q \cdot 1 \rceil \\ &= \lfloor \log_2 e \rfloor + \lceil \log_2 \sum_{0 \leq j < \lambda} e_j (2^1)^{j-(\lambda-1)} \rceil \end{aligned}$$

steps in parallel. If $e = 2^{\lambda-1}$ then $A = 0$ and $\delta_{2,(1,1)}(e) \geq \lfloor \log_2 e \rfloor = \lambda_2(e) - 1 = \lceil \log_2 e \rceil$. Otherwise $A = 1$, which yields $\delta_{2,(1,1)}(e) \geq \lfloor \log_2 e \rfloor + 1 = \lambda_2(e) = \lceil \log_2 e \rceil$ as already stated in Corollary 4.21. If $c_Q = 0$ and $c_A = 1$ then we have

$$\delta_{q,(0,1)}(e) \geq 0 \cdot Q + 1 \cdot A = \lceil \log_2 [e]_{q,0} \rceil = \lceil \log_2 \sum_{0 \leq j < \lambda} e_j \rceil = \lceil \log_2 \omega_q(e) \rceil$$

if $A \geq 1$. The formula also holds for $A = 0$, and therefore Corollary 4.24 can be derived from Result 4.17. This shows that Result 4.17 covers both the optimal bound by Kung (1976) for original addition chains, and the one by von zur Gathen (1991) for addition chains with free scalar.

4.3. Consequence to parallel exponentiation. We have shown that both the upper and the lower bound depend on the quotient $c = \frac{c_Q}{c_A}$, i.e. the ratio between the cost for a q-step and for a non-q-step. Both statements—the algorithmic approach of Corollary 4.1 and the lower bound in Result 4.17—identify the cost $c_Q \cdot (\lambda_q(e) - 1)$ to be the bottleneck. We compare upper and lower bound in more detail now, and we want to extract consequences to parallel exponentiation. Both bounds are identical for original addition chains and q-addition chains with free scalar. Therefore, our estimate on the possible speed-up is given by optimal bounds. To illustrate the case $0 < c < 1$ for 2-addition chains, we have done some experiments on the remaining three sets of weight that have been introduced in Section 3.4.

Original addition chains. The optimal original addition chain that computes an exponent $e \in \mathbb{N}_{\geq 1}$ in parallel has depth exactly $\delta_{2,(1,1)}(e) = \lceil \log_2 e \rceil$. An (often too generous) upper bound on a sequential addition chain for e is given by the binary addition chain; Corollary 3.15 states $\ell_2(e) \leq \lambda_2(e) + \nu_2(e) - 2 \leq 2\lfloor \log_2 e \rfloor$. Thus, the maximal speed-up is limited by

$$\frac{\ell_2(e)}{\delta_{2,(1,1)}(e)} \leq \frac{\lambda_2(e) + \nu_2(e) - 2}{\lceil \log_2 e \rceil}$$
$$\begin{cases} = \frac{\lfloor \log_2 e \rfloor + 1 + 1 - 2}{\lceil \log_2 e \rceil} = \frac{\log_2 e}{\log_2 e} = 1 & \text{if } e = 2^{\lambda_2(e)-1}, \\ < \frac{\lambda_2(e) + \nu_2(e)}{\lambda_2(e)} = 1 + \frac{\nu_2(e)}{\lambda_2(e)} \leq 2 & \text{else.} \end{cases}$$

Since we expect $\nu_2(e)$ to be roughly $\frac{1}{2}\lambda_2(e)$ on average, the expected speed-up compared to the *binary addition chain* is only $\frac{3}{2}$. If we compare the optimal depth of the parallel addition chain to *Brauer addition chains*, then we have an even more disappointing result. We have $\ell_2(e) \in \lambda_2(e) + O(\frac{\lambda_2(e)}{\log_2 \lambda_2(e)})$ by (3.19), and therefore

$$\lim_{e \to \infty} \frac{\ell_2(e)}{\delta_{2,(1,1)}(e)} = 1.$$

Thus, no benefit can be taken from parallelism.

Free scalar. For q-addition chains with weight $c_A \cdot (0, 1)$ the situation changes. Algorithm 4.14—which is the algorithm described by von zur Gathen (1991)—takes advantage of free q-steps; Corollary 4.24 states an optimal depth of $\delta_{q,(0,1)} = \lceil \log_2 \omega_q(e) \rceil$ for an exponent $e \in \mathbb{N}_{\geq 1}$ and a scalar $q \in \mathbb{N}_{\geq 2}$. Other authors, e.g.

Agnew *et al.* (1988) and Stinson (1990), have also discussed algorithmic ideas for parallel exponentiation. They restrict to $\mathbb{F}_{2^n}$ and the special case of free squarings. In the case when a free scalar can be assumed, we hope for a better speed-up. For a finite field $\mathbb{F}_{q^n}$, we have $\omega_q(e) \leq \sum_{0 \leq i < \lambda_q(e)} (q-1) \leq (q-1) \cdot n$. By Corollary 3.10, any power of an element of $\mathbb{F}_{q^n}$ can be computed in $O(\frac{n(q-1)}{\log n + \log (q-1)})$ multiplications in $\mathbb{F}_{q^n}$ whenever raising to a q-th power is free. Compared to this sequential algorithm, we have depth at most $\delta_{q,(0,1)}(e) \leq \lceil \log_2(q-1) + \log_2 n \rceil$. Thus, we expect a speed-up of $O(\frac{n(q-1)}{\log^2 (n(q-1))})$, and a high potential for massive parallelism.

The remaining case. It remains to discuss 2-addition chains with weight (c_Q, c_A) such that $0 < c < 1$. The upper bound of Theorem 4.13 does not match the lower bound of Result 4.17 in this case. We have done some calculations on those three sets of weights of Section 3.4 where c is neither 0 nor 1 to show that both estimates are nevertheless close to each other.

The goal of these calculations is two-fold. On the one hand, the values should reflect the quality of Algorithm 4.12 compared to the lower bound. On the other hand, we give concrete estimates on the maximal possible speed-ups for these three sets of weights. These three sets are the same sets as in Section 3.4. They are inspired by different constructions of the finite field $\mathbb{F}_{2^n}$ using a polynomial basis representation, see Section 5.2. We label the set with weights roughly $(2, 3)$ by *Arbitrary*; see Table A.6. The two other sets have decreasing weights; the set *Sparse* is stated in Table A.7, the set *Sedimentary* in Table A.8.

First, we discuss the quality of Algorithm 4.12. The estimates on the depth are given for the worst case in column 8 of Tables A.6–A.8. In the worst case, all digits of the binary representation of the exponent e are non-zero. Thus $e = \sum_{0 \leq i < n} 1 \cdot 2^i = 2^n - 1$ for a n-bit exponent. Column 9 shows the difference[23] between this upper bound and the lower bound on $e = 2^n - 1$. It is 1 for set *Arbitrary* with weight $(2, 3)$, and at most 2 for the two other sets *Sparse* and *Sedimentary*[24]. This gap does not depend on the binary length n of the exponent nor on the choice of $c = \frac{c_Q}{c_A}$. Thus, the depth of Algorithm 4.12 is close to the optimal depth in the worst case. This is also true for all other n-bit exponents $2^{n-1} \leq e < 2^n$. Column 5 lists the lower bound for the best case $e = 2^{n-1}$. This bound is sharp since 2^{n-1} can be computed with $n-1$ many doublings. The lower bound for all other n-bit exponents is in the interval bounded by 2^{n-1} and $2^n - 1$. For the set *Arbitrary* the gap between both numbers (column 7) is only 1. In the two other cases, the size of the interval seems to depend logarithmically on the reciprocal value of c. The maximal gap between the lower bound for an exponent $2^{n-1} \leq e < 2^n$ and depth of Algorithm 4.12 is given in column 9. This validates

[23] Tables A.6–A.8 list the normalized depth L/c_A for all entries.

[24] Since the lengths are normalized, 1 stands for the depth in which one non-q-step with cost c_A can be computed.

that our algorithmic approach is close to be optimal for the three sets. We have

Figure 4.10: Comparison of Algorithm 4.12 for three out of the five sets of weights introduced in Section 3.4. We give the depth relative to the average length of the binary addition chain. Thus this figure gives the expected speed-up of the parallel algorithm related to the repeated squaring algorithm for an arbitrary exponent.

computed the speed-up of Algorithm 4.12 relative to the average length of the binary addition chain, i.e. the normalized length $L/c_A = \frac{1}{2}(n-1) + c \cdot (n-1)$. This average length (column 4 of Tables A.6–A.8) meets the average number of steps given by the experiment of Section 3.4. The quotient of the average length to the depth of Algorithm 4.12 can be interpreted as the expected speed-up if we choose a n-bit exponent at random. Figure 4.10 shows that we cannot expect a high speed-up for a fixed weight with $c \approx \frac{2}{3}$. The reason is that addition chains with weight $(2,3)$ are close to original addition chains. The predicted speed-up is roughly 1.75 (see Table A.6, column 10), compared to 1.5 for original addition chains. Algorithm 4.12 can be executed on 3 processors in parallel for the set *Arbitrary*.

For the sets *Sparse* and *Sedimentary*, the situation is different. The speed-up increases from 1.98 for $n = 209$ up to 17.40 for $n = 9998$ for *Sparse*. This is caused by decreasing values for c. For the set *Sedimentary* the speed-up is at least 1.84 for $n = 209$ and at most 14.14 for $n = 9998$. The graph of the speed-up is illustrated in Figure 4.10. The fluctuations are caused by the non-monotonous decrease of c, see column 3 of Tables A.7–A.8. Algorithm 4.12 can be performed on at most 35 and 28 processors for *Sparse* and *Sedimentary*, respectively. The calculations

predict that exponentiation benefits from parallel computing for these sets of weights.

Provisional conclusions. The above discussion suggests to use the quotient $c = \frac{c_Q}{c_A}$ as the main parameter to choose a good data structure for parallel exponentiation.

From now on, we will work in the finite field $\mathbb{F}_{q^n}$. Beside multiplications, which are related to non-q-steps, we focus on the Frobenius automorphism to evaluate the q-th power for different basis representations. The Frobenius automorphism originally has motivated the introduction of q-steps. We aim to find a data structure for parallel exponentiation which beats sequential exponentiation significantly.

5. Polynomial bases

Our main perspective is a data structure that supports parallel exponentiation in a finite field $\mathbb{F}_{q^n}$. Up to now we have discussed exponentiation in the model of weighted addition chains with scalar. In Section 4 we have characterized a suitable data structure for exponentiation by the ratio $c = \frac{c_Q}{c_A}$. Now we apply the previous results on parallel exponentiation to certain basis representations of finite fields. In this section we start with the polynomial basis representation of $\mathbb{F}_{q^n}$. We investigate how different moduli influence the performance of the basic arithmetic operations. Most of the material covered here can be found in von zur Gathen & Nöcker (2000).

This section is organized as follows: In Section 5.1 we give an outline on polynomial multiplication as the basic operation for this basis representation. It can be skipped if the reader is familiar with fast polynomial multiplication. The arithmetic in $\mathbb{F}_q[x]/(f)$ is presented in Section 5.2. In conformity with weighted q-addition chains we focus on multiplications, i.e. non-q-steps, and the computation of the Frobenius automorphism, i.e. q-steps. We contrast an arbitrary choice of the modulus f (Section 5.2.1) with two different types of sparse moduli. Beside trinomials and pentanomials (Section 5.2.2)—which are the preferred moduli in the recently passed IEEE Standard Specifications for Public-Key Cryptography[25]—we analyze sedimentary polynomials (Section 5.2.3) as modulus. The latter one were suggested by Coppersmith (1984). Each subsection includes reports on experiments on the arithmetic in $\mathbb{F}_{2^n}$. We have chosen extensions over the binary field for our experiments; these are of special practical interest, see also IEEE (2000). The conclusions on parallel exponentiation are given in Section 5.3; a sparse modulus offers significant improvements compared to an arbitrary modulus when representing the finite field by a polynomial basis. For completeness the closing Section 5.4 summarizes the basic facts on division in a polynomial basis representation of $\mathbb{F}_{q^n}$.

5.1. Polynomial arithmetic. The standard way to represent (and implement) the finite field $\mathbb{F}_{q^n}$ with q^n elements is by a *polynomial basis representation.*

FACT 5.1 (Lidl & Niederreiter 1983, Theorem 1.61). *For a $f \in \mathbb{F}_q[x]$, the residue ring $\mathbb{F}_q[x]/(f)$ is a field if and only if f is irreducible over $\mathbb{F}_q$.*

There always exists an irreducible polynomial f over $\mathbb{F}_q$ of degree $n \in \mathbb{N}_{\geq 1}$. Then $\mathbb{F}_q[x]/(f)$ is a field with q^n elements. It may be regarded as a vector space over $\mathbb{F}_q$ with basis $\mathcal{B} = ((1 \bmod f), (x \bmod f), \ldots, (x^{n-1} \bmod f))$. We call such a basis a

[25] "The reduction of polynomials modulo $p(t)$ is particularly efficient if $p(t)$ has a small number of terms. [...] Thus, it is a common practice to choose a trinomial for the field polynomial, provided that one exists. If an irreducible trinomial of degree m does not exist, then the next best polynomials are the *pentanomials* [...].", IEEE (2000), A.3.4, p. 80.

polynomial basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$. An element $A \in \mathbb{F}_q[x]/(f)$ is given by a $\mathbb{F}_q$-linear combination $A = \sum_{0 \le i < n} A_i x^i \bmod f$ with $A_0, \ldots, A_{n-1} \in \mathbb{F}_q$. We may identify an element $A \in \mathbb{F}_q[x]/(f)$ with the polynomial $g = \sum_{0 \le i < n} g_i x^i \in \mathbb{F}_q[x]$ of degree at most $n-1$, where $g_i = A_i$ for all $0 \le i < n$. This polynomial is called the *canonical representative*.

Thus, we can perform arithmetic in $\mathbb{F}_q[x]/(f)$ by doing polynomial arithmetic in $\mathbb{F}_q[x]$. Addition of the canonical representatives $g = \sum_{0 \le i < n} g_i x^i$ and $h = \sum_{0 \le i < n} h_i x^i \in \mathbb{F}_q[x]$ can be done with at most n additions in $\mathbb{F}_q$. The resulting polynomial $v = \sum_{0 \le i < n} (g_i + h_i) x^i$ has degree less than n, i.e. v is the canonical representative. Multiplication of two polynomials of degree at most $n-1$ yields a product polynomial of degree at most $2(n-1)$. Since we want to use the canonical representative, a final reduction modulo f has to be performed whenever an arbitrary representative has degree at least $\deg f = n$. We will see below that this reduction is the crucial point for fast exponentiation in a polynomial basis representation. But first we look at some algorithms for polynomial multiplication since fast polynomial multiplication is at the core of fast polynomial arithmetic. Algorithms for division, inversion and many other basic arithmetic tasks can be reformulated to profit from fast multiplication. The book of von zur Gathen & Gerhard (1999) illustrates this deep relation in an impressing way. We refer to Section 8 of their book for all details that are omitted in the following sketch.

We call a function $\mathsf{M}\colon \mathbb{N}_{>0} \to \mathbb{R}_{>0}$ a *multiplication time*[26] for $\mathbb{F}_q[x]$ if two polynomials in $\mathbb{F}_q[x]$ of degree less than n can be multiplied using at most $\mathsf{M}(n)$ operations in $\mathbb{F}_q$.

The classical multiplication algorithm. Let g and h be polynomials over $\mathbb{F}_q$ with $g = \sum_{0 \le i < m} g_i x^i$ and $h = \sum_{0 \le i < n} h_i x^i$. The *classical multiplication algorithm* computes the coefficients $u_0, \ldots, u_{n+m-2}$ of the product $u = g \cdot h$ by straightforward computation:

$$u = \sum_{0 \le i \le (n-1)+(m-1)} u_i x^i = \left(\sum_{0 \le i < m} g_i x^i \right) \cdot \left(\sum_{0 \le i < n} h_i x^i \right)$$

$$= \sum_{0 \le i \le (n-1)+(m-1)} \left(\sum_{j \in \mathcal{I}_i} g_j \cdot h_{i-j} \right) x^i$$

where $\mathcal{I}_i = \{0 \le j \le m\colon 0 \le i - j < n\}$ is the set of indices for the i-th coefficient. The product can be computed with at most $m \cdot n$ multiplications and at most $(m-1) \cdot (n-1)$ additions in $\mathbb{F}_q$.

[26] This is taken from Definition 8.26 in von zur Gathen & Gerhard (1999).

REMARK 5.2. *Two polynomials of degree $m-1$ and $n-1$, respectively, can be multiplied with $m \cdot n$ multiplications and $(m-1) \cdot (n-1)$ additions in $\mathbb{F}_q$. Thus, we have*

$$\mathsf{M}(n) \leq 2n^2 - 2n + 1 \in O(n^2).$$

Karatsuba's algorithm. Now let both polynomials g and h be of degree less than $n \in \mathbb{N}_{\geq 1}$. We may rewrite the classical multiplication algorithm by applying a *divide-and-conquer* strategy. We write

$$g = G_1 x^m + G_0 \qquad \text{and} \qquad h = H_1 x^m + H_0$$

with $G_1, G_0, H_1, H_0 \in \mathbb{F}_q[x]$ of degree less than $m = \lceil \frac{n}{2} \rceil$. Then

$$\begin{aligned} g \cdot h &= (G_1 x^m + G_0) \cdot (H_1 x^m + H_0) \\ &= (G_1 \cdot H_1) x^{2m} + (G_1 \cdot H_0 + G_0 \cdot H_1)\, x^m + G_0 \cdot H_0. \end{aligned} \tag{5.3}$$

Karatsuba used a trick—which is originally described for large integer multiplication in Karatsuba & Ofman (1963)—to reduce the number of polynomial multiplication in (5.3) from 4 to 3. He observed that

$$G_1 \cdot H_0 + G_0 \cdot H_1 = (G_1 + G_0) \cdot (H_1 + H_0) - G_1 \cdot H_1 - G_0 \cdot H_0. \tag{5.4}$$

By (5.3), this leads to the following algorithm.

ALGORITHM 5.5. Karatsuba's multiplication algorithm.

Input: Two polynomials $g, h \in \mathbb{F}_q[x]$ of degree less than $n \in \mathbb{N}_{\geq 1}$.
Output: The product polynomial $u = g \cdot h \in \mathbb{F}_q[x]$.

1. If $n = 1$ then compute $u \leftarrow g \cdot h$ in $\mathbb{F}_q$.
2. Else
3. Let $m = \lceil \frac{n}{2} \rceil$ and write $g = G_1 x^m + G_0$ and $h = H_1 x^m + H_0$ with $G_0, G_1, H_0, H_1 \in \mathbb{F}_q[x]$ of degree less than m.
4. Call the algorithm recursively to compute $U_0 \leftarrow G_0 \cdot H_0$ and $U_2 \leftarrow G_1 \cdot H_1$ and $U_1 \leftarrow (G_1 + G_0) \cdot (H_1 + H_0)$.
5. Set $u = U_2 x^{2m} + U_0$ and compute $U_1 \leftarrow U_1 - U_2 - U_0$.
6. Compute $u \leftarrow u + U_1 x^m$.
7. Return u.

LEMMA 5.6. *The algorithm computes the product of two polynomials in $\mathbb{F}_q[x]$ of degree less than 2^k, where $k \in \mathbb{N}_{\geq 0}$, with at most $9 \cdot (2^k)^{\log_2 3} - 8 \cdot 2^k$ operations in $\mathbb{F}_q$.*

PROOF. Correctness of Algorithm 5.5 follows directly by (5.3) and (5.4). We prove the bound $T(2^k) \leq 9 \cdot (2^k)^{\log_2 3} - 8 \cdot 2^k$ on the operations in $\mathbb{F}_q$ for the multiplication of two polynomials of degree less than 2^k by induction on $k \in \mathbb{N}_{\geq 0}$. For $k = 0$ we have $T(2^0) = 1 = 9 \cdot (2^0)^{\log_2 3} - 8 \cdot 2^0$ multiplication in step 1. Thus, we suppose that the claim is true for $0 \leq k' < k$. Since $m = \lceil \frac{2^k}{2} \rceil = 2^{k-1}$ we have $2 \cdot 2^{k-1}$ additions in $\mathbb{F}_q$ and three recursive calls of input size 2^{k-1} in step 4. The latter counts for $3T(2^{k-1})$ operations in $\mathbb{F}_q$. Since $\deg U_0 < 2 \cdot 2^{k-1} = 2m$, only the computation of U_1 causes operations in $\mathbb{F}_q$ in step 5; we count at most $2 \cdot 2^k$ additions of coefficients. Finally, step 6 causes 2^k further additions in $\mathbb{F}_q$. Summing up, we have $T(2^k) \leq 3T(2^{k-1}) + 4 \cdot 2^k$. By the induction hypothesis, this is just

$$\begin{aligned} T(2^k) &\leq 3 \cdot \left(9 \cdot (2^{k-1})^{\log_2 3} - 8 \cdot 2^{k-1}\right) + 4 \cdot 2^k = 3 \cdot 9 \cdot 3^{k-1} - 3 \cdot 4 \cdot 2^k + 4 \cdot 2^k \\ &= 9 \cdot 3^k - 2 \cdot 4 \cdot 2^k = 9 \cdot (2^k)^{\log_2 3} - 8 \cdot 2^k \end{aligned}$$

as claimed. □

COROLLARY 5.7. *Two polynomials in $\mathbb{F}_q[x]$ of degree less than $n \in \mathbb{N}_{\geq 1}$ can be multiplied with at most*

$$\mathsf{M}(n) \leq 27 n^{\log_2 3} - 8n \in O(n^{1.59})$$

operations in $\mathbb{F}_q$.

PROOF. This follows directly from Lemma 5.6 using the fact that $n \leq 2^k < 2n$ for $k = \lceil \log_2 n \rceil$. □

Nearly linear multiplication algorithms. The (asymptotic) fastest multiplication algorithm known so far uses the *Fast Fourier Transformation*. It would exceed the scope of this work to sketch the main ideas of this fast multiplication algorithm. Therefore, we refer to the book of von zur Gathen & Gerhard (1999). The basics, including the *Discrete Fourier Transformation*, are explained in their Section 8.2. The application to polynomial multiplication are investigated in the subsequent Section 8.3. Originally, Schönhage & Strassen (1971) presented the algorithm for integer multiplication. Schönhage (1977) gave the extension to polynomial multiplication over fields of characteristic 2. We cite the version given in von zur Gathen & Gerhard (1999), Theorem 8.23, but we restrict to polynomials over a finite field $\mathbb{F}_q$.

FACT 5.8 (Schönhage & Strassen 1971, Schönhage 1977). *Two univariate polynomials with coefficient in $\mathbb{F}_q$ of degree less than n in can be multiplied using at most*

$$\mathsf{M}(n) \in O(n \log n \operatorname{loglog} n)$$

operations in $\mathbb{F}_q$.

This is nearly linear in n. To emphasize this, one may use the *soft-Oh-notation* $O^{\sim}$, see von zur Gathen & Gerhard (1999), Section 25.7. It swallows the log-factors such that $\mathsf{M}(n) \in O^{\sim}(n)$ which is also referred to as being *softly linear*.

Cantor (1989) described another algorithm which computes the product with only $O^{\sim}(n)$ operations in $\mathbb{F}_q$. His original work had been restricted to the multiplication of two polynomials over a prime field; von zur Gathen & Gerhard (1996) gave a detailed analysis in the general case. They also discussed in detail simplifications for polynomial multiplication over the binary field $\mathbb{F}_2$ in Section 3 of the more extended technical report. Indeed, Jürgen Gerhard has implemented this algorithm in the software library BiPolAr, see von zur Gathen & Gerhard (1996), Section 7 and von zur Gathen & Gerhard (1999), Section 9.7 for descriptions. We cite their estimate on the cost; a detailed proof can be found in their technical report.

Fact 5.9 (von zur Gathen & Gerhard 1996, Theorem 2.9). *Two polynomials in $\mathbb{F}_{q^m}[x]$ with product of degree less than n, where $q \leq n \leq q^m$, can be multiplied using less than $O((\frac{q}{\log q})^2 \cdot n \log^2 n)$ operations in $\mathbb{F}_{q^m}$.*

As described in the technical report related to von zur Gathen & Gerhard (1996), we can apply this method to multiply two polynomials g and h over the field $\mathbb{F}_q$ with $\deg(g \cdot h) < n$ as follows. We first choose an extension field $\mathbb{F}_{q^m} \cong \mathbb{F}_q[x]/(u)$ with u irreducible of degree $\deg u = m$ such that $n \leq \frac{m}{2} \cdot q^m$. We rewrite the factors g and h as polynomials of the form $g = \sum_{0 \leq i < n'} G_i x^{m'i}$ and $h = \sum_{0 \leq i < n'} H_i x^{m'i}$ with $m' = \lceil \frac{m}{2} \rceil$ and $n' = \lceil \frac{n}{m'} \rceil$. Here $\deg G_i$ and $\deg H_i$ are less than m' for all $0 \leq i < n'$, and G_i and H_i may be regarded as elements in $\mathbb{F}_{q^m}$. The coefficients of the product polynomial of g and h over $\mathbb{F}_{q^m}$ have degree less than m. By construction, the coefficients of the product polynomial $g \cdot h$ in $\mathbb{F}_q$ can be uniquely determined from the product polynomial in $\mathbb{F}_{q^m}[x]$. Applying classical multiplication and division with remainder to perform the arithmetic in $\mathbb{F}_{q^m}$ gives the following estimate.

Fact 5.10 (von zur Gathen & Gerhard 1996). *Two polynomials of degree less than $n \in \mathbb{N}_{\geq 1}$ in $\mathbb{F}_q[x]$ can be multiplied with at most*

$$\mathsf{M}(n) \leq O(\frac{q^2}{\log^3 q} \cdot n \log^3 n)$$

operations in $\mathbb{F}_q$ providing that $q < n$.

5.2. Computing the canonical representative. We now consider the problem of computing the canonical representative $v = g \operatorname{rem} f$ of an element $B = (g \bmod f)$ in the field $\mathbb{F}_q[x]/(f)$. By definition the canonical representative v satisfies $\deg v < \deg f = n$.

5.2.1. Arbitrary modulus. We start with the case where the modulus $f = x^n + \sum_{0 \le i < n} f_i x^i \in \mathbb{F}_q[x]$ is an arbitrary monic irreducible polynomial. For given positive integers n and q—with q a prime power—such an irreducible polynomial can be constructed within an expected number of $O(n^2 \log^2 n \operatorname{loglog} n \cdot \log q)$ operations in $\mathbb{F}_q$, see Ben-Or (1981), Theorem 3. The main task is to compute the canonical representative $v = g \operatorname{rem} f$ of an element $B = (g \bmod f) \in \mathbb{F}_{q^n}$. The basic algorithm for this is *division with remainder*; it computes the two uniquely determined polynomials $u, v \in \mathbb{F}_q[x]$ for $f, g \in \mathbb{F}_q[x]$ with $f \neq 0$ such that $g = u \cdot f + v$ and v is either 0 or $\deg v < \deg f = n$. Algorithm 5.19 is a slightly modified version.

FACT 5.11. *Division with remainder of a polynomial g of degree $m \ge n$ by a monic polynomial f of degree n can be done with at most $2n(m-n+1) \in O(mn)$ operations in $\mathbb{F}_q$.*

The product of two canonical representatives in $\mathbb{F}_q[x]/(f)$ has degree at most $2(n-1)$. Multiplying two elements in $\mathbb{F}_q[x]/(f)$ and determing the canonical representative of this product can thus be done with $\mathsf{M}(n) + O(n^2) \in O(n^2)$ operations in $\mathbb{F}_q$. But we can improve on the number of operations.

Division with remainder using Newton iteration. We want to profit from fast polynomial multiplication when computing the canonical representative. This can be done using the idea of *reversals*. The following is along von zur Gathen & Gerhard (1999), Section 9.1; see also Section 8.3 in Aho *et al.* (1974).

We define the *reversal* $\operatorname{rev}_m(g)$ *of a polynomial* $g = \sum_{0 \le i \le m} g_i x^i \in \mathbb{F}_q[x]$ by

$$\operatorname{rev}_m(g) = x^m g\left(\frac{1}{x}\right) = \sum_{0 \le i \le m} g_{m-i} x^i,$$

that is, we revert the order of the coefficients of g. Then the equation $g = u \cdot f + v$ with $\deg f = n$ and $\deg g = m \ge n$ can be written as

$$\begin{aligned} \operatorname{rev}_m(g) &= x^m \cdot g\left(\frac{1}{x}\right) = x^m \cdot (uf + v)\left(\frac{1}{x}\right) \\ &= x^{m-n} \cdot u\left(\frac{1}{x}\right) \cdot x^n \cdot f\left(\frac{1}{x}\right) + x^{m-n+1} \cdot x^{n-1} \cdot v\left(\frac{1}{x}\right) \\ &= \operatorname{rev}_{m-n}(u) \cdot \operatorname{rev}_n(f) + x^{m-n+1} \cdot \operatorname{rev}_{n-1}(v) \end{aligned}$$

We get ride of the reversal of v by computing this equation modulo x^{m-n+1}, i.e.

$$\operatorname{rev}_m(g) \equiv \operatorname{rev}_{m-n}(u) \cdot \operatorname{rev}_n(f) \bmod x^{m-n+1}. \tag{5.12}$$

Since f has degree n, the absolute coefficient of $\mathrm{rev}_n(f)$ is not zero. Thus, $\mathrm{rev}_n(f)$ is invertible modulo x^{m-n+1}. Then

$$\mathrm{rev}_{m-n}(u) \equiv \mathrm{rev}_m(g) \cdot \mathrm{rev}_n(f)^{-1} \bmod x^{m-n+1}. \tag{5.13}$$

and we have $u = \mathrm{rev}_{m-n}(\mathrm{rev}_{m-n}(u))$ and $v = g - u \cdot f$. We get the following algorithm where we suppose the inverse $\mathrm{rev}_n(f)^{-1} \bmod x^{m-n+1}$ as an input.

ALGORITHM 5.14. Fast division with remainder.

Input: Polynomials $f \in \mathbb{F}_q[x]$ of degree n and $g \in \mathbb{F}_q[x]$ of degree $m \geq n$, respectively, and the inverse of the reversal of f modulo x^{m-n+1}.

Output: Uniquely determined $u, v \in \mathbb{F}_q[x]$ such that $g = u \cdot f + v$ with $\deg v < n$.

1. Compute $w \leftarrow \mathrm{rev}_m(g) \cdot \mathrm{rev}_n(f)^{-1} \operatorname{rem} x^{m-n+1}$.
2. Set $u = \mathrm{rev}_{n-m}(w)$.
3. Compute $v \leftarrow g - u \cdot f$.
4. Return u and v.

LEMMA 5.15. *Algorithm 5.14 performs division with remainder of a polynomial g of degree $m \geq n$ by a polynomial f of degree n with at most $\mathsf{M}(m) + \mathsf{M}(n) + n$ operations in $\mathbb{F}_q$ if $\mathrm{rev}_n(f)^{-1} \bmod x^{m-n+1}$ is given.*

PROOF. Correctness of Algorithm 5.14 follows by (5.12) and (5.13). For the estimate on the costs we observe that $\deg \mathrm{rev}_n(f)^{-1} \bmod x^{m-n+1}$ has degree at most $m - n$. The multiplication in step 1 is therefore a multiplication of two polynomials of degree less than m with costs $\mathsf{M}(m)$. In step 3, the algorithm only has to compute the lower degree part of $g - u \cdot f$, since $\deg v < n$. This can be done with $\mathsf{M}(n) + n$ operations in $\mathbb{F}_q$. □

With the Extended Euclidean Algorithm 5.29 we can compute $\mathrm{rev}_n(f)^{-1} \bmod x^{m-n+1}$ using $O(mn)$ operations. A faster algorithm is presented in von zur Gathen & Gerhard (1999), Theorem 9.4. They use *Newton iteration* which computes the inverse of a polynomial modulo x^{m-n+1} in at most $3\mathsf{M}(m-n+1)+m-n+1 \in O(\mathsf{M}(m-n+1))$ operations in $\mathbb{F}_q$.

Field arithmetic. It remains to conclude the costs for multiplication, i.e. c_A, and for computing the Frobenius automorphism, i.e. c_Q, in $\mathbb{F}_q[x]/(f)$ if we use the idea of reversals to calculate the canonical representatives.

COROLLARY 5.16. *Let $\mathbb{F}_{q^n}$ be given by a polynomial basis representation with monic irreducible modulus $f = x^n + \sum_{0 \leq i < n} f_i x^i \in \mathbb{F}_q[x]$. Furthermore, let $\mathrm{rev}_n(f)^{-1} \bmod x^{n-1}$ be given. Then two elements can be multiplied with at most $3\mathsf{M}(n) + n$ operations in $\mathbb{F}_q$. An element can be raised to the q-th power with at most $(3\mathsf{M}(n) + n) \cdot \ell_2(q)$ operations in $\mathbb{F}_q$, where $\ell_2(q) \leq 2\lfloor \log_2 q \rfloor$ is the minimal length of an (original) addition chain for q.*

PROOF. The costs for the multiplication of two elements of $\mathbb{F}_q[x]/(f)$ is a direct consequence of Lemma 5.15; the product of two representatives has degree $m \leq 2(n-1)$. A q-th power can be computed by applying an addition chain for q of minimal length $\ell_2(q)$. Each of the $\ell_2(q)$ steps is a multiplication modulo f. □

In the special case $q = 2$, we can compute $g^2 = (\sum_{0 \leq i \leq m} g_i x^i)^2 = \sum_{0 \leq i \leq m} g_i x^{2i}$ without operations in $\mathbb{F}_2$ since the *Frobenius automorphism* is a linear map due to (2.8). Thus, only the reduction modulo f causes operations in $\mathbb{F}_2$.

COROLLARY 5.17. *Let $\mathbb{F}_{2^n}$ be represented by $\mathbb{F}_2[x]/(f)$, and let $\mathrm{rev}_n(f)^{-1} \bmod x^{n-1}$ be given. Then two elements in $\mathbb{F}_{2^n}$ can be multiplied with $3\mathsf{M}(n) + n$ operations in $\mathbb{F}_2$. Squaring can be done with $2\mathsf{M}(n) + n$ operations in $\mathbb{F}_2$.*

The reduction uses about $2\mathsf{M}(n)+n$ operations of the total number of operations. Hence, at most 2/3 of the computation time for multiplication and almost all time for squaring is spent for the reduction modulo f.

Experiments. We did the following experiments with arbitrary modulus $f = x^n + \sum_{0 \leq i < n} f_i x^i \in \mathbb{F}_2[x]$. The test series *Arbitrary* consists of 50 values for the extension degree $n \in \mathbb{N}_{\geq 1}$ with $n \approx 200 \cdot i$ for $1 \leq i \leq 50$, see also Table A.9, column 1. For all 50 test values n we computed an arbitrary irreducible polynomial f of degree $\deg f = n$ over the binary field $\mathbb{F}_2$ using the subroutine `randIrrpoly1`[27] of BIPOLAR.

We precomputed and stored $\mathrm{rev}_n(f)^{-1} \bmod x^{n-1}$ for each f; the documented times do not include the times of this precomputation. For each n of test series *Arbitrary* we ran 10000 trials for both multiplication and squaring modulo f using BIPOLAR's basic arithmetic methods as described in von zur Gathen & Gerhard (1999), Section 9.7. The times for both multiplication and squaring are approximated by the average of the 10000 trials. For multiplication, we generated two polynomials of degree less than n at random for each trial. The time for one trial includes the multiplication of the two polynomials and the subsequent reduction modulo the irreducible polynomial f. For $n \leq 4000$ BIPOLAR performs reduction by classical division with remainder. For larger values of n, reduction via Newton iteration becomes faster. Therefore, the library computes the canonical representative according to Algorithm 5.14 for $n > 4000$. Squaring is done by expanding a randomly chosen polynomial. This causes no operations in $\mathbb{F}_2$. As for multiplication, the times include only this expansion and the subsequent reduction.

Corollary 5.17 states that we can expect an asymptotic ratio of $\frac{2}{3}$ between squaring and multiplication time in $\mathbb{F}_{2^n}$. Figure 5.1 exactly shows this ratio for degrees $n > 4000$, see the blue line which is connected to the right y-axis. The

[27] This subroutine computes a random irreducible polynomial of input degree n by trying to factor successively randomly chosen polynomials. The basic algorithm is the distinct degree factorization; see von zur Gathen & Gerhard (1999) for an introduction on factorization.

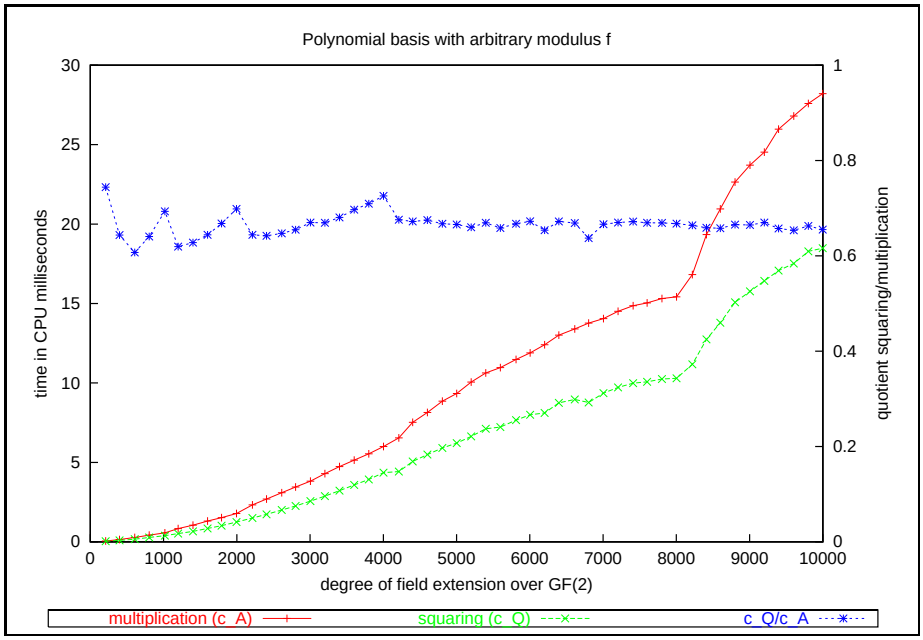


Figure 5.1: Times for multiplication (+) and squaring (×) for the test series *Arbitrary*. The times are the average of 10000 trials. Here $\mathbb{F}_{2^n}$ is given by a polynomial basis representation due to $\mathbb{F}_2[x]/(f)$ with $f \in \mathbb{F}_2[x]$ an arbitrary irreducible polynomial. The times are marked on the left y-axis. The scale of the right y-axis relates to the quotient (∗) between both times.

wobbling line for test points $n \leq 4000$ appears because BiPolAr uses Karatsuba multiplication for $n > 500$ but sticks to classical division with remainder which is faster in this range. The times for multiplication and squaring agree not only by the ratio but also if we compare them to one single polynomial multiplication of two random polynomials of degree less than n. Columns 3 and 5 of Table A.9 show the times relative to the times for one single polynomial multiplication. These so-called *normalized times* are close to 3 and 2, respectively, as predicted by Corollary 5.17. Therefore, this data structure—polynomial basis representation with general modulus—is a candidate with poor speed-up for parallel exponentiation.

5.2.2. Sparse polynomials. By the above, the crucial point for arithmetic in $\mathbb{F}_q[x]/(f)$ with an arbitrary chosen modulus f is the reduction step. To circumvent this bottleneck, we now analyze irreducible polynomials which support the reduction, see also the remark in Ben-Or (1981)[28]. Some authors, e.g. Schroeppel

[28] "In order to make residue computation mod $g(x)$ easier one looks for special types of irreducible polynomials such as $g(x) = x^n + x + a$, $a \in \mathbb{Z}_p$.", Ben-Or (1981), p. 395. [$g(x)$ is f in our notation.]

et al. (1995a)[29] and De Win *et al.* (1996)[30], suggested to speed up arithmetic by choosing a special kind of a *sparse irreducible polynomial* f as modulus, in particular a *trinomial*. A *sparse polynomial* f, in the usual sense, is a polynomial of the form

$$f = \sum_{1 \le i \le t} f_i x^{e_i} \in \mathbb{F}_q[x]$$

with coefficients $f_1, \ldots, f_t \in \mathbb{F}_q$ and integers $0 \le e_1 < e_2 < \ldots < e_t = \deg f$ such that $t \ll \deg f = n$. We suppose that f is monic, i.e. $f_t = 1$.

EXAMPLE 5.18. (i) A *binomial* has only 2 non-zero coefficients and is of type $f = x^n + f_1 x^{e_1}$. For all irreducible binomials we have $e_1 = 0$. The only irreducible binomial over $\mathbb{F}_2$ is $x + 1$ since otherwise $x^n + 1$ is divisible by $x + 1$ for $n \in \mathbb{N}_{\ge 2}$.

(ii) A sparse polynomial with exactly 3 non-zero coefficients is called a *trinomial*. An irreducible trinomial over $\mathbb{F}_2$ has the form $f = x^n + x^k + 1 \in \mathbb{F}_q[x]$ with $0 < k < n$.

(iii) We call a polynomial $f = x^n + f_4 x^{e_4} + f_3 x^{e_3} + f_2 x^{e_2} + f_1 x^{e_1} \in \mathbb{F}_q[x]$ with 5 non-zero coefficients a *pentanomial*.

(iv) A sparse polynomial of the form $f = x^n + h \in \mathbb{F}_q[x]$ with $s = \deg h \ll n$ is called *sedimentary*. Here all non-zero coefficients but the leading one are at the bottom of f, and $s = \deg h \ge t - 2$. ◊

To define $\mathbb{F}_{q^n}$ by a sparse irreducible polynomial, the following numbers are helpful. Let q be a prime power and n a positive integer, set

$$\tau_q(n) = \min \left\{ t \in \mathbb{N}_{>0} : \begin{array}{l} \text{there exists a sparse irreducible polynomial in} \\ \mathbb{F}_q[x] \text{ of degree } n \text{ with } t \text{ non-zero coefficients} \end{array} \right\} \text{ and}$$

$$\theta_q(n) = \min \{ s = \deg h \in \mathbb{N}_{\ge 0} : x^n + h \in \mathbb{F}_q[x] \text{ is irreducible} \}.$$

Obviously, $\tau_q(n) - 2 \le \theta_q(n)$ since sedimentary polynomials are a special kind of sparse polynomials. For all pairs q and n there exists an irreducible monic polynomial f in $\mathbb{F}_q[x]$ of degree n, and both $\tau_q(n)$ and $\theta_q(n)$ are well-defined with $\theta_q(n) \le n - 1$.

[29] "The irreducible trinomial $T(u)$ has a structure that makes it a pleasant choice for representing the field.", Schroeppel *et al.* (1995a), p. 4.

[30] "We will show that the reduction operation can be speeded up even further if an irreducible *trinomial* is used [...].", De Win *et al.* (1996), Section 3.

Division with remainder for sparse polynomials. As already mentioned, we hope for a substantial speed-up if we use sparse polynomials as modulus. This expected improvement is caused by two observations. For all sparse polynomials, the number of non-zero coefficients in f is much less than the degree n of f. For sedimentary polynomials, the gap $\Delta = n - e_{t-1} - 1 = n - s - 1$ between the leading coefficient and the second non-zero coefficient is large. We adapt the division with remainder to both special cases. For practical terms, we can take advantage of modern word-oriented processors with 32 or even 64 bit per word, because the enclosed zeros support word-wise computation. This is often much faster than the bit-wise computation.

The following is a tailor-made version of the well-known division with remainder to make the most of a sparse divisor.

ALGORITHM 5.19. Division with remainder for sparse polynomials.

Input: $n \in \mathbb{N}_{\geq 1}$, a polynomial $g \in \mathbb{F}_q[x]$ of degree m, an integer $t \geq 2$, positive integers $0 \leq e_1 < e_2 < \ldots < e_t = n$, and coefficients $f_1, \ldots, f_t \in \mathbb{F}_q$ with $f_t = 1$.

Output: Uniquely determined polynomials u and v in $\mathbb{F}_q[x]$ such that $g = u \cdot \left(\sum_{1 \leq i \leq t} f_i x^{e_i}\right) + v$ with $\deg v < n$.

1. Set $j = m - n$ and $v = g$ and $u = 0$.
2. While $j \geq 0$ do 3–7
3. Set $\Delta = \min\{n - e_{t-1} - 1, j\} \geq 0$. *Comment:* the maximal number of coefficients in v which can be handled at once is $\Delta + 1$.
4. Set $w = \sum_{0 \leq i \leq \Delta} v_{n+j-i} x^i$. *Comment:* the top-degree part of u is also the top-degree part of v.
5. Compute $v \leftarrow v - w \cdot \left(\sum_{1 \leq i < t} f_i x^{e_i}\right) x^{j-\Delta}$.
6. Set $v \leftarrow v - w x^{n+j-\Delta}$ and $u \leftarrow u + w x^{j-\Delta}$.
7. Set $j \leftarrow j - (\Delta + 1)$.
8. Return (u, v).

LEMMA 5.20. *The algorithm performs division with remainder of a polynomial $g \in \mathbb{F}_q[x]$ of degree $m \geq n$ by $f = x^n + \sum_{1 \leq i < t} f_i x^{e_i} \in \mathbb{F}_q[x]$ in at most $2(t-1)(m-n+1)$ operations in $\mathbb{F}_q$.*

PROOF. To prove correctness as well as the bound on the number of operations, we formulate and show the following invariant for $f = \sum_{1 \leq i \leq t} f_i x^{e_i}$ for the loop in the steps 2–7:

$$g = u \cdot f + v \quad \text{and} \quad \deg v \leq n + j \quad \text{and} \quad x^{j+1} \text{ divides } u.$$

Before the first lap of the loop, we have $u = 0$ and $v = g$ and $j = m - n$ in step 3. Then, $u \cdot f + v = 0 \cdot f + g = g$ and $\deg v = \deg g = m = n + j$ and x^{j+1} divides

$u = 0$. Thus, we suppose that the invariant is true for a fixed j before a new lap. Let u' and v' be the polynomials after step 7. Then we have

$$\begin{aligned} u'f + v' &\overset{5.,6.}{=} (u + wx^{j-\Delta})f + (v - wx^{n+j-\Delta} - w \cdot \sum_{1 \le i < t} f_i x^{e_i} x^{j-\Delta}) \\ &= (uf + v) + wx^{j-\Delta}(f - (x^n + \sum_{1 \le i < t} f_i x^{e_i})) \\ &= g + wx^{j-\Delta} \cdot 0 = g. \end{aligned}$$

By step 7 we have $j' = j - (\Delta + 1)$, i.e. the decremented value of the counter j. The polynomial w has degree at most Δ, and thus $u' = u - w \cdot x^{j-\Delta}$ is divisible by $\min\{x^{j+1}, x^{j-\Delta}\} = x^{j+1-(\Delta+1)} = x^{j'+1}$ since $j - \Delta < j + 1$. The first $\Delta + 1$ coefficients of v are subtracted to get v', hence $\deg v' \le \deg v - (\Delta + 1) \le n + j - (\Delta + 1) = n + j'$. The invariant holds, and Algorithm 5.19 is correct: After the loop, we have $uf + v = g$ and $\deg v \le n + (-1) < n$.

For the costs we remark that w in step 4 can be extracted from v without operations in $\mathbb{F}_q$. The invariant also shows that the manipulation in step 6 is without any operations in $\mathbb{F}_q$. In step 5, we can successively compute $w \cdot f_i x^{e_i+j-\Delta}$ using $\Delta + 1$ scalar multiplications and further $\Delta + 1$ additions for all $1 \le i < t$. Thus, one lap causes $2(\Delta+1)(t-1)$ operations in $\mathbb{F}_q$. We have a total of $S \in \mathbb{N}_{\ge 1}$ laps with $m - n + 1 - S \cdot (\Delta + 1) \le 0 < m - n + 1 - (S - 1)(\Delta + 1)$, that is $S = \lceil \frac{m-n+1}{n-e_{t-1}} \rceil$. For $S-1$ laps, we have $2(\Delta+1)(t-1) = 2(n-e_{t-1})(t-1)$ operations in $\mathbb{F}_q$ for each lap. For the final, lap we have $\Delta = m - n - (S - 1)(n - e_{t-1})$, and we count $2 \cdot (m - n - (S - 1)(n - e_{t-1}) + 1)(t - 1)$ operations in $\mathbb{F}_q$ in Step 5. This yields a total of

$$\begin{aligned} & (S-1) \cdot 2(n - e_{t-1})(t-1) + 2(m - n - (S-1)(n - e_{t-1}) + 1)(t-1) \\ = \; & (S-1) \cdot 2(n - e_{t-1})(t-1) + 2(m - n + 1)(t-1) \\ & - 2(S-1)(n - e_{t-1})(t-1) \\ = \; & 2(m - n + 1)(t-1) \end{aligned}$$

operations as claimed. □

Field arithmetic. As for arbitrary modulus we summarize the previous results with respect to arithmetic in $\mathbb{F}_{q^n}$ if the finite field is defined by a sparse irreducible polynomial.

COROLLARY 5.21. *Let $t = \tau_q(n)$ and $\mathbb{F}_{q^n}$ be given by a polynomial basis representation with sparse modulus $f = x^n + \sum_{1 \le i < t} f_i x^{e_i}$. Then two elements can be multiplied with at most $\mathsf{M}(n) + 2(t-1)(n-1)$ operations in $\mathbb{F}_q$. An element can be raised to the q-th power with at most $2(q-1)(t-1)(n-1)$ or $\mathsf{M}(n)\ell_2(q) + 2(t-1)(n-1)\ell_2(q)$ operations in $\mathbb{F}_q$, where $\ell_2(q) \le 2\lfloor \log_2(q) \rfloor$ is the minimal length for an (original) addition chain for q.*

PROOF. The bounds are a direct consequence of Lemma 5.20 because the product of two canonical representatives of elements in $\mathbb{F}_q[x]/(f)$ has degree $m \leq 2(n-1)$. A q-th power of a polynomial over $\mathbb{F}_q$ can be calculated without field operations. The result has degree $m \leq q \cdot (n-1)$ and has to be reduced, which gives the first bound. One may alternatively apply an addition chain for q of minimal length $\ell_2(q)$, and we get the second bound. □

For $q = 2$ we have $\ell_2(2) = 1$, and thus a square can be calculated with at most $2(t-1)(n-1)$ operations in $\mathbb{F}_{2^n}$. The arithmetic suggested by the IEEE Standard 1363 gives in the following bounds.

COROLLARY 5.22. *Let $\mathbb{F}_{2^n} \cong \mathbb{F}_2[x]/(f)$ be defined by an irreducible trinomial or pentanomial, respectively. Then two elements in $\mathbb{F}_{2^n}$ can be multiplied in at most*

$\mathsf{M}(n) + 4n - 4$ *operations if f is a trinomial, and*
$\mathsf{M}(n) + 8n - 8$ *operations if f is a pentanomial.*

Squaring can be done with $4n-4$ and $8n-8$ operations in $\mathbb{F}_2$, respectively.

Experiments. We ran a test series *Sparse* with irreducible trinomials and pentanomials in $\mathbb{F}_2[x]$ for the same 50 values of n as in test series *Arbitrary*. In 26 cases there exist irreducible trinomials of the required degree. For all other values of n an irreducible pentanomial is used as modulus. Again, we used BIPOLAR's irreducibility test but adapted it to irreducible trinomials. In the subsequent paragraph, we explain the basic algorithmic ideas in the subroutine `findIrrTrinom` of BIPOLAR.

For the arithmetic we extended BIPOLAR implementing two versions of Algorithm 5.19: one for trinomials (subroutine `t3_remainder`) and one for general sparse modulus (subroutine `sparse_remainder`[31]). We supplemented BIPOLAR's division routine `precremainder` by a switch to make sure that division by a sparse modulus works according to Algorithm 5.19.

We repeated both the trials for multiplication and for squaring 10000 times. The only difference to the test series *Arbitrary* is that the reduction is now by the sparse irreducible polynomials of the test series *Sparse*. The times which are the average of all trials are given in Table A.10; they are illustrated in Figure 5.2. Theory and experiment both show that the times for multiplication in $\mathbb{F}_{2^n}$ are now close to times for a polynomial multiplication, see column 4 of Table A.10. Squaring is as fast as expected, see the green line in Figure 5.2. The relation $\frac{c_Q}{c_A}$ shows the decrease that we have expected for $\mathsf{M}(n^{1.59})$. We observe that a single multiplication is nearly 3 times as fast as in the case of arbitrary modulus. Thus, we gain both faster arithmetic and a better speed-up when we use a sparse modulus to represent $\mathbb{F}_{2^n}$ instead of an arbitrary one.

[31] The version for general sparse modulus was gratefully implemented by Sandor Eckar.

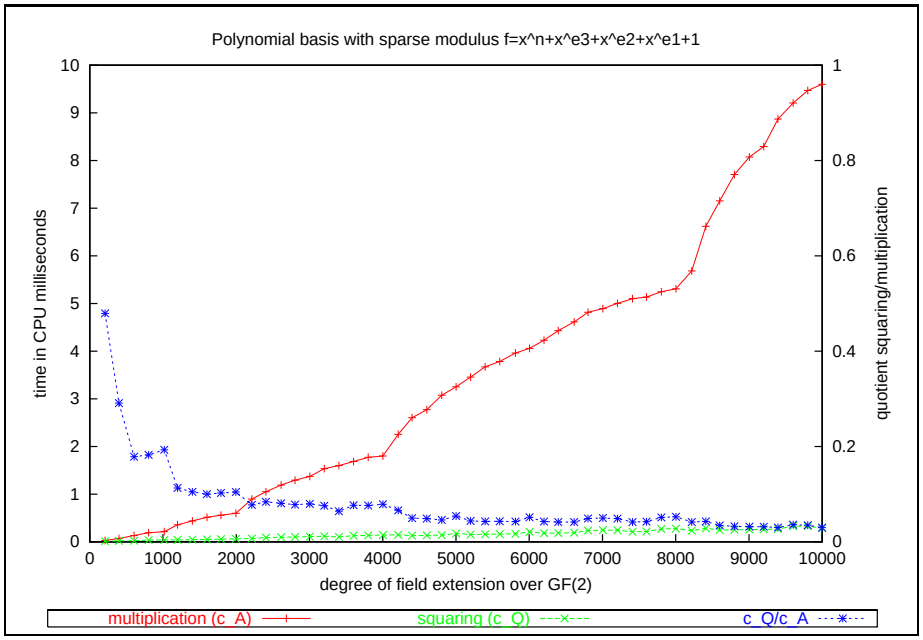


Figure 5.2: Times for multiplication (+) and squaring (×) for the test series *Sparse*. The times are the average of 10000 trials. Here $\mathbb{F}_{2^n}$ is given by a polynomial basis representation due to $\mathbb{F}_2[x]/(f)$ with $f \in \mathbb{F}_2[x]$ an irreducible polynomial with at most $t = 5$ non-zero coefficients. The times are marked on the left y-axis. The scale of the right y-axis relates to the quotient (∗) between both times.

Finding irreducible sparse polynomials. As discussed above, constructing a finite field $\mathbb{F}_{q^n} \cong \mathbb{F}_q[x]/(f)$ by an irreducible trinomial $f = x^n + f_2 x^{e_2} + f_1 \in \mathbb{F}_q[x]$ supports efficient arithmetic in $\mathbb{F}_{q^n}$. We have already cited some remarks of Ben-Or (1981), Schroeppel *et al.* (1995a), and De Win *et al.* (1996). For completeness, we sketch the ideas we used to find an irreducible trinomial $x^n + x^k + 1$ over $\mathbb{F}_2$. We follow Zierler & Brillhart (1968, 1969) in our approach to find irreducible trinomials. A similar idea is suggested in IEEE (2000), A.8.2 and A.8.5.

An obvious idea is to test all trinomials $x^n + x^k + 1 \in \mathbb{F}_2[x]$, for given $n \in \mathbb{N}_{\geq 2}$ and $1 \leq k < n$, successively for irreducibility until an irreducible one appears—or all candidates are proven to be reducible.

Ree (1971) considers trinomials of type $x^n + x + a \in \mathbb{F}_q[x]$, where $n \in \mathbb{N}_{\geq 2}$ and q is a prime not dividing $2n(n-1)$. In his Theorem 1 he proved that there are roughly $\frac{q}{n}$ irreducible trinomials of this type in $\mathbb{F}_q[x]$. Wassermann (1993), Satz 4.4.2, generalized this result to monic trinomials $x^n + f_2 x + f_1 \in \mathbb{F}_q[x]$, where $f_1 \in \mathbb{F}_q^\times$ is fixed, q is a prime power and $\mathrm{char}(\mathbb{F}_q)$ does not divide $2n(n-1)$.

Ben-Or (1981) gave an algorithm to decide whether a polynomial $f \in \mathbb{F}_q[x]$ is irreducible or not, using at most $O(n\mathsf{M}(n)\log(nq))$ operations in $\mathbb{F}_q$; see von zur Gathen & Gerhard (1999), Section 14.9, for a detailed presentation. Thus, we

can find an irreducible trinomial $x^n+x^k+1 \in \mathbb{F}_2[x]$ with at most $O(n^2\mathsf{M}(n)\log n)$ operations in $\mathbb{F}_2$. Motivated by the results of Ree (1971) and Wassermann (1993) on special trinomial cited above and our experimental observations, we assume that randomly chosen trinomials factor like arbitrary polynomials. Then we could expect a total of only $O(n\mathsf{M}(n)\log^2 n)$ operations in $\mathbb{F}_2$ to find an irreducible trinomial over $\mathbb{F}_2$ of degree n, if one exists.

We can speed up our searching by restricting the number of trinomials $x^n + x^k + 1 \in \mathbb{F}_2[x]$ that have to be tested for irreducibility by Ben-Or's algorithm. The following fact is useful.

FACT 5.23 (Golomb 1967). *The polynomial $f \in \mathbb{F}_q[x]$ is irreducible if and only if its reversal $\mathrm{rev}_n(f) \in \mathbb{F}_q[x]$ is irreducible.*[32]

Therefore, we can restrict to the case $1 \le k \le \lfloor \frac{n}{2} \rfloor$ to check for the existence of a trinomial of degree $n \in \mathbb{N}_{\ge 2}$. Furthermore we can use a sieve by Swan (1962).

FACT 5.24 (Swan 1962, Corollary 5). *Let $n > k > 0$ be integers. Assume exactly one of n and k is odd. Then $x^n + x^k + 1 \in \mathbb{F}_2[x]$ has an even number of factors (and hence is reducible) in the following cases:*

(i) n is even, k is odd, $n \neq 2k$ and $\frac{nk}{2} \equiv 0$ or $1 \bmod 4$, or

(ii) n is odd, k is even, k does not divide $2n$, and $n \equiv \pm 3 \bmod 8$, or

(iii) n is odd, k is even, k divides $2n$, and $n \equiv \pm 1 \bmod 8$.

In all other cases $x^n + x^k + 1$ has an odd number of factors over $\mathbb{F}_2$.

If n and k are both odd, we can also apply Swan's result as he already mentioned himself[33]. In practice, Fact 5.24 speeds up the search for irreducible trinomials over $\mathbb{F}_2$ significantly. Using this strategy, we found an irreducible trinomial for given $2 \le n < 10000$ in 5146 cases using BIPOLAR's factorization routine `findIrrTrinom`. The division by the tested trinomial has been computed with the subroutine `t3_remainder`. For the remaining 4852 field extensions $\mathbb{F}_{2^n}$ there exist no irreducible trinomials, but there are irreducible pentanomials. Thus, we conjecture that $\tau_2(n) \le 5$ for all $n \in \mathbb{N}_{\ge 2}$.

[32] "Since the transformation $f(x) \to x^n f(1/x)$ changes any polynomial of degree n into its 'reverse', [...] and does the same for each of its factors, it is sufficient to list only those trinomials with $a \le [n/2]$.", Golomb (1967), p. 90–91.

[33] "The case where n and k are both odd can be reduced to the case k even by considering $x^n + x^{n-k} + 1$ which has the same number of irreducible factors as $x^n + x^k + 1$.", Swan (1962), p. 1106.

5.2.3. Sedimentary polynomials. Another type of sparse polynomials relevant to fast arithmetic in $\mathbb{F}_q[x]/(f)$ are sedimentary polynomials $f = x^n + h$ with $s = \deg h \ll n$. These are special types of sparse polynomials with $s \leq t + 2$. Heuristically, one might hope that $\theta_q(n) = \min\{\deg h \colon x^n + h \in \mathbb{F}_q[x] \text{ irreducible}\}$ is roughly $\log_q n$ since there are about n polynomials with degree up to $\log_q n$, and a fraction of about $1/n$ of all polynomials of fixed degree n is irreducible[34] in $\mathbb{F}_q[x]$. Similar ideas were expressed by Coppersmith (1984)[35] and Gordon & McCurley (1992)[36]. Indeed, we found irreducible polynomials $f = x^n + h$ with $s = \deg h \leq \lceil \log_2 n \rceil + 1 \ll n$ for all tested n in order to construct the finite fields of the test series *Sedimentary*. The experiments of Gao & Panario (1997)[37] and Gao *et al.* (1999)[38] checked $\theta_2(n) \leq 3 + \log_2 n$ for $n \leq 2000$. Our experiments showed $\theta_2(n) \leq 3 + \lceil \log_2 n \rceil$ for $n \leq 8142$.

Arithmetic for sedimentary modulus. Obviously, we can rewrite the results for sparse polynomials in the case of sedimentary polynomials using $t \leq s + 2$. But we may also profit from the fact that $n - e_{t-1} = n - s$ is large. Then in Algorithm 5.19 step 5, we may multiply the two polynomials w and $h = f - x^n$ directly, performing a fast algorithm for multiplication in $\mathbb{F}_q[x]$.

LEMMA 5.25. *Let $f, g \in \mathbb{F}_q[x]$ be two polynomials with $\deg g = m \geq n$ and $f = x^n + h$ sedimentary such that $0 \leq s = \deg h \leq \frac{n-1}{2}$. Division with remainder of g by f can be performed with at most $\left(\frac{m-n+1}{s+1} + 1\right) \mathsf{M}(s+1) + 2(m-n+1) + s$ operations in $\mathbb{F}_q$.*

PROOF. We can compute division with remainder using Algorithm 5.19 with minimal modifications. Since we want to profit from the sequence of zeros between x^n and h, the new step 3 is now:

3'. Set $\Delta = \min\{s, j\}$.

[34] Precise bounds can be found in Lidl & Niederreiter (1983), Exercises 3.26 and 3.27: The number $N_q(n)$ of irreducible monic polynomials in $\mathbb{F}_q[x]$ of degree n is bounded by $\frac{1}{n}q^n - \frac{q}{n(q-1)}(q^{n/2} - 1) \leq N_q(n) \leq (1/n)(q^n - q)$.

[35] "Choose a primitive polynomial $P(x)$ of degree n, such that $P(x) = x^n + Q(x)$, where the degree of $Q(x)$ is smaller than $n^{2/3}$. (This should be possible; heuristically, for a given n, we expect the best possible $Q(x)$ to have degree about $\log_2 n$.)", Coppersmith (1984), p. 590.

[36] "[...] it is convenient that we construct our finite field $GF(2^n)$ as $GF(2)[x]/(f(x))$, where f is an irreducible polynomial of the form $x^n + f_1(x)$, with f_1 of small degree. [...] a search that we made confirms this since it is possible to find an f_1 of degree at most 11 for all n up to 600, and it is usually possible to find one of degree at most 7.", Gordon & McCurley (1992), p. 313.

[37] "Experimental results show that such polynomial f exists for $n \leq 1000$ taking $\deg g \leq 2 + \log_2 n$.", Gao & Panario (1997), p. 358. [The g is our h.]

[38] "We also did experiments on the existence of irreducible polynomials of the form $x^n + g(x) \in \mathbb{F}_q[x]$ with $\deg g(x) = \log n + O(1)$. For $q = 2$ and $n \leq 2000$, it turns out that such irreducible polynomials always exist with $\deg g(x) \leq \log n + 3$.", Gao *et al.* (1999), p. 49. [The g is our h.]

By assumption, $s \leq \frac{n-1}{2}$ which gives $n - s - 1 \geq s$. Hence, the chosen Δ is at most the one of the original algorithm. Obviously, this does not interfere with the correctness.

In step 5, we can then perform fast polynomial multiplication on the two polynomials w of degree $\Delta \leq s$ and $h = f - x^n = \sum_{1 \leq i < t} f_i x^{e_i}$ of degree (at most) s. This can be done with $\mathsf{M}(s+1)$ operations in $\mathbb{F}_q$. We count at most $s + 1 + \Delta$ further additions in $\mathbb{F}_q$ to perform $v - w \cdot h \cdot x^{j-\Delta}$.

A lap of Algorithm 5.19 step 2–7 decreases the degree of v by $\Delta + 1$. There are $S = \lceil \frac{m-n+1}{s+1} \rceil$ many laps. Analogous to the proof of Lemma 5.20, we have to distinguish between $S - 1$ laps with $\Delta = s$ and a final lap with $\Delta = m - n - (S-1)(s+1)$. Each of the S laps contains one multiplication $w \cdot h$ with at most $\mathsf{M}(s+1)$ operations in $\mathbb{F}_q$. Hence, the total costs are

$$\begin{aligned} & S \cdot \mathsf{M}(s+1) + (S-1)(2s+1) + s + 1 + m - n - (S-1)(s+1) \\ = \; & S \cdot \mathsf{M}(s+1) + (S-1) \cdot s + m - n + 1 + s \\ \leq \; & S \cdot \mathsf{M}(s+1) + 2 \cdot (m - n + 1) + s. \end{aligned}$$

Here we have inserted $S = \lceil \frac{m-n+1}{s+1} \rceil \leq \frac{m-n+1}{s+1} + 1$. □

Field arithmetic. For arithmetic in $\mathbb{F}_{q^n}$, we have $m \leq 2(n-1)$ and $m - n + 1 \leq n - 1$. The next two corollaries follow directly from this and Lemma 5.25.

COROLLARY 5.26. *Let $\mathbb{F}_{q^n}$ be given by a polynomial basis representation with sedimentary modulus $f = x^n + h$ such that $\deg h = s \leq (n-1)/2$. Then two elements can be multiplied with at most $\mathsf{M}(n) + \left(\frac{n-1}{s+1} + 1\right) \mathsf{M}(s+1) + 2(n-1) + s$ operations in $\mathbb{F}_q$. An element can be raised to the q-th power with at most $\mathsf{M}(n)\ell_2(q) + \left(\left(\frac{n-1}{s+1} + 1\right) \mathsf{M}(s+1) + 2(n-1) + s\right) \ell_2(q)$ or $\left(\frac{(q-1)(n-1)}{s+1} + 1\right) \mathsf{M}(s+1) + 2(q-1)(n-1) + s$ operations in $\mathbb{F}_q$ where $\ell_2(q)$ is the minimal length for an (original) addition chain for q with $\ell_2(q) \leq 2\lfloor \log_2 q \rfloor$.*

COROLLARY 5.27. *Let $\mathbb{F}_{2^n}$ be given by $\mathbb{F}_2[x]/(f)$ where f is an irreducible sedimentary polynomial $f = x^n + h$ of $\deg h \in O(\log_2 n)$. Then two elements in $\mathbb{F}_{2^n}$ can be multiplied with at most*

$$\mathsf{M}(n) + O\left(\frac{n}{\log n} \mathsf{M}(\log n)\right) \in \mathsf{M}(n) + O(n \operatorname{loglog} n \cdot \operatorname{logloglog} n),$$

squaring can be done with $O(\frac{n}{\log n} \mathsf{M}(\log n)) \in O(n \operatorname{loglog} n \cdot \operatorname{logloglog} n)$ operations in $\mathbb{F}_2$.

Experiments. We took a third test series *Sedimentary* for a polynomial basis representation of $\mathbb{F}_{2^n}$. This test series consists of irreducible sedimentary polynomials $f = x^n + h \in \mathbb{F}_2[x]$ of the same degrees n as in *Arbitrary*. We chose the

degree of the sediment as small as possible; the degree of h is given in Table A.11, column 2. Indeed, all sediments h have degree at most $\log_2 n + 2$. The version of Algorithm 5.19 for sedimentary polynomials was implemented and added to BiPolAr as subroutine `s_remainder` by Olaf Müller. Again a switch makes sure that division modulo a sedimentary polynomial is done by this new subroutine whenever routine `precremainder` is called for a sedimentary polynomial. The experiment itself is just a repetition of the previously described for *Arbitrary* and *Sparse*. We only exchanged the division routine by the tailor-made one for sedimentary polynomials. The times are documented in Figure 5.3 and Table A.11.

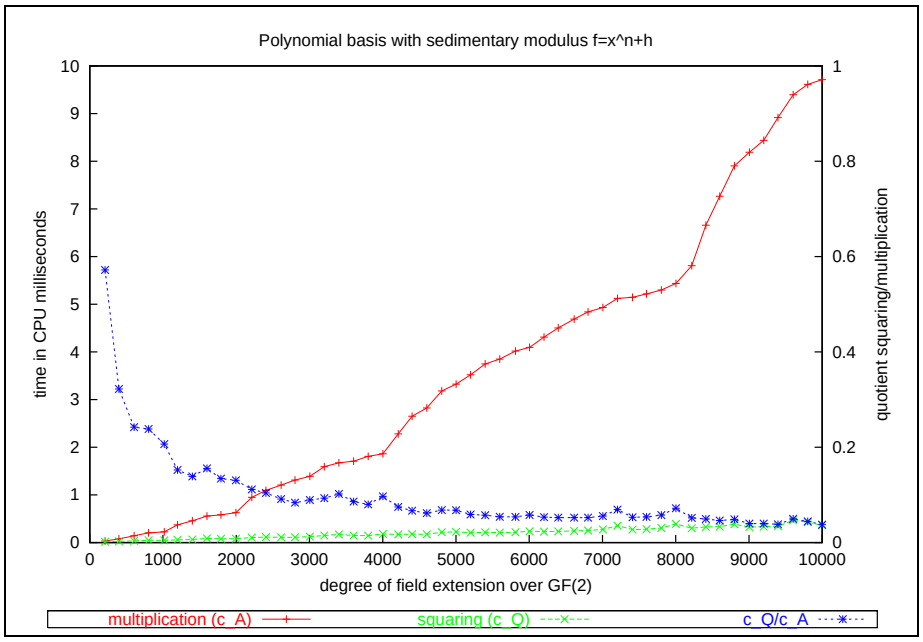


Figure 5.3: Times for multiplication (+) and squaring (×) for the test series *Sedimentary*. The times are the average of 10000 trials. Here $\mathbb{F}_{2^n}$ is given by a polynomial basis representation due to $\mathbb{F}_2[x]/(f)$ with $f = x^n + h \in \mathbb{F}_2[x]$ an irreducible polynomial of $\deg h \leq \lceil \log_2 n \rceil + 1$. The times are marked on the left y-axis. The scale of the right y-axis relates to the quotient (∗) between both times.

As for *Sparse*, we observe that the reduction is now very cheap. Multiplication is dominated by the cost for multiplying two canonical representatives, and squaring decreases if we compare it to multiplication, see Table A.11, columns 4 and 7. This is in conformity with the theoretical results of Corollary 5.27. Generally speaking, the results for sparse and for sedimentary irreducible polynomials are close to each other in theory as well as by experiment. Thus, we expect a similar behavior of both data structures.

5.3. Exponentiation in a polynomial basis representation. We summarize the findings on arithmetic in $\mathbb{F}_{q^n}$ if the field is represented by a polynomial

basis, i.e. $\mathbb{F}_q[x]/(f)$. In particular, it remains to apply these facts to (parallel) exponentiation. For all three different kind of moduli described above we give a table with three columns showing the results for classical arithmetic (Remark 5.2) with $\mathsf{M}(n) = O(n^2)$, Karatsuba's multiplication algorithm (Corollary 5.7) with $\mathsf{M}(n) = O(n^{1.59})$, and the fastest known FFT-based multiplication (Fact 5.8) with $\mathsf{M}(n) = O(n \log n \cdot \operatorname{loglog} n)$. We restrict to the case $q = 2$.

Polynomial basis representation: arbitrary modulus			
	classical	Karatsuba	FFT
arithmetic			
multiplication (c_A)	$4n^2 - 4n + 1$	$O(n^{1.59})$	$O(n \log n \cdot \operatorname{loglog} n)$
squaring (c_Q)	$2n^2 - 2n + 1$	$O(n^{1.59})$	$O(n \log n \cdot \operatorname{loglog} n)$
$c = \frac{c_Q}{c_A}$	$\approx \frac{1}{2}$	$\approx \frac{2}{3}$	$\approx \frac{2}{3}$
exponentiation			
sequential	$2n^3 + O(\frac{n^3}{\log n})$	$O(n^{2.59})$	$O(n^2 \log n \cdot \operatorname{loglog} n)$
parallel	$2n^3 + O(n^2 \log n)$	$O(n^{2.59})$	$O(n^2 \log n \cdot \operatorname{loglog} n)$

Table 5.4: Bounds on exponentiation in $\mathbb{F}_{2^n}$ given by a polynomial basis representation with arbitrary modulus.

For arbitrary modulus the result is disappointing, see Table 5.4. Independent of the chosen multiplication algorithm, exponentiation shows no real impact by parallel computing. The asymptotic bounds of Corollary 3.10 for sequential and Corollary 4.1 for parallel exponentiation do not differ. We conclude with the help of this bounds that a polynomial basis representation, with the modulus $f \in \mathbb{F}_2[x]$ taken as an arbitrary chosen irreducible polynomial, is not a good data structure for parallel exponentiation in $\mathbb{F}_{2^n}$.

For sparse modulus the situation is much better, see Table 5.5. For classical multiplication the bounds on exponentiation drops from $O(\frac{n^3}{\log n})$ to soft-quadratic. If multiplication becomes faster the speed-up using parallel computing decreases. But the asymptotic bounds are quadratic in this case. Thus, we can expect clearly better results using a sparse instead of an arbitrary irreducible polynomial f to construct $\mathbb{F}_{2^n}$.

The bounds for a sedimentary irreducible polynomial $f = x^n + h \in \mathbb{F}_2[x]$ are displayed in Table 5.6. The entries are slightly higher than those in Table 5.5 for sparse moduli. The reason is that the asymptotic bound for squaring for sedimentary modulus is only soft-linear, while it is linear for sparse modulus. If we restrict to a polynomial basis representation of $\mathbb{F}_{2^n}$, our choice would be a sparse irreducible trinomial or pentanomial.

5.4. Division in a polynomial basis representation. We have not integrated subtraction as a basic operation in our model of q-addition chains. A

Polynomial basis representation: sparse modulus ($\tau_2(n) \leq 5$)			
	classical	Karatsuba	FFT
arithmetic			
mult. (c_A)	$2n^2 + 6n - 7$	$O(n^{1.59})$	$O(n \log n \cdot \operatorname{loglog} n)$
squaring (c_Q)	$8n - 8$	$8n - 8$	$8n - 8$
$c = \frac{c_Q}{c_A}$	$O(\frac{1}{n})$	$O(\frac{1}{n^{\log_2 3}})$	$O(\frac{1}{\log n \cdot \operatorname{loglog} n})$
exponentiation			
sequential	$O(\frac{n^3}{\log n})$	$O(\frac{n^{2.59}}{\log n})$	$O(n^2 \operatorname{loglog} n)$
parallel	$2n^2 \log_2 n$ $+O(n^2)$	$8n^2$ $+O(n^{1.59} \log n)$	$8n^2$ $+O(n \log^2 n \cdot \operatorname{loglog} n)$

Table 5.5: Bounds on exponentiation in $\mathbb{F}_{2^n}$ given by a polynomial basis representation with sparse modulus.

Polynomial basis representation: sedimentary modulus ($\theta_2(n) \in O(\log n)$)			
	classical	Karatsuba	FFT
arithmetic			
multiplication (c_A)	$2n^2 + O(n \log n)$	$O(n^{1.59})$	$O(nL(n))$
squaring (c_Q)	$O(n \log n)$	$O(n \log^{0.59} n)$	$O(nLn)$
$c = \frac{c_Q}{c_A}$	$O(\frac{\log n}{n})$	$O((\frac{\log n}{n})^{\log_2 3})$	$O(\frac{\operatorname{logloglog} n}{\log n})$
exponentiation			
sequential	$O(\frac{n^3}{\log n})$	$O(\frac{n^{2.59}}{\log n})$	$O(n^2 L(\log n))$
parallel	$O(n^2 \log n)$	$O(n^2 \log^{0.59} n)$	$O(n^2 L(\log n))$

Table 5.6: Bounds on exponentiation in $\mathbb{F}_{2^n}$ given by a polynomial basis representation with sedimentary modulus. Here $L(n) = \log n \cdot \operatorname{loglog} n$.

subtraction step would correspond to a division in $\mathbb{F}_{q^n}$. Nevertheless, we shortly sketch the algorithm to perform division in $\mathbb{F}_q[x]/(f)$.

Let $A \in \mathbb{F}_{q^n}^{\times} = \mathbb{F}_{q^n} \setminus \{0\}$ be a field element. As above we assume f to be an irreducible polynomial over $\mathbb{F}_q$ and all elements to be represented by polynomials over $\mathbb{F}_q$ of degree less than $n = \deg f$. Let g be the canonical representative of A. We can substitute division by A by multiplication with the inverse $A^{-1} \in \mathbb{F}_{q^n}$. The canonical representative h of A^{-1} satisfies $g \cdot h \equiv 1 \bmod f$. We can solve the following *linear Diophantine equation* for g and f to get a representative for A^{-1}: Find $u, h \in \mathbb{F}_q[x]$ such that

$$f \cdot u + g \cdot h = 1.$$

The Extended Euclidean Algorithm. The common way to compute u and h in $\mathbb{F}_q[x]$—or to prove that such polynomials do not exist—uses the *Extended*

Euclidean Algorithm. The book of von zur Gathen & Gerhard (1999) gives an extensive presentation of this algorithm. We only collect a few facts from their book and adapt them to invert elements in $\mathbb{F}_{q^n}$.

Actually, the Extended Euclidean Algorithm computes the greatest common divisor of two elements in a ring $\mathcal{R}$ whenever division with remainder is available in $\mathcal{R}$. Such a ring is also called an *Euclidean domain.*

DEFINITION 5.28. *Let $\mathcal{R}$ be a ring and $A, B, C \in \mathcal{R}$. Then C is the* greatest common divisor *(or* gcd *for short) of A and B if C divides both A and B, and C is also divisible by all common divisors of A and B.*

This definition does not ensure uniqueness. To get an unique gcd in the case $\mathcal{R} = \mathbb{F}_q[x]$, we impose the gcd to be monic, i.e. the leading coefficient is supposed to be 1. We give the algorithm in the case $\mathcal{R} = \mathbb{F}_q[x]$.

EXTENDED EUCLIDEAN ALGORITHM 5.29.

Input: Polynomials $f, g \in \mathbb{F}_q[x]$ with $\deg f = n \geq m = \deg g$.
Output: Polynomials $d, u, h \in \mathbb{F}_q[x]$ such that $d = \gcd(f, g) = u \cdot f + h \cdot g$.

1. Set $a_0 = f$, $a_1 = g$, $u_0 = 1$, $u_1 = 0$, $v_0 = 0$, and $v_1 = 1$.
2. Set $i = 1$.
3. While $a_i \neq 0$ do 4–6
4. Compute w_i and a_{i+1} such that $a_{i-1} = w_i \cdot a_i + a_{i+1}$ by division with remainder.
5. Compute $u_{i+1} \leftarrow u_{i-1} - w_i \cdot u_i$ and $v_{i+1} \leftarrow v_{i-1} - w_i \cdot v_i$.
6. Set $i \leftarrow i + 1$.
7. Set $\ell = i - 1$ and Return a_ℓ, u_ℓ, v_ℓ.

FACT 5.30 (von zur Gathen & Gerhard 1999, Theorem 3.11). *Let $f, g \in \mathbb{F}_q[x]$ be polynomials of degree $n = \deg f$ and $m = \deg g$, respectively. The Extended Euclidean Algorithm 5.29 computes $d, s, t \in \mathbb{F}_q[x]$ such that $d = \gcd(f, g) = s \cdot f + t \cdot g$ with at most $O(nm)$ operations in $\mathbb{F}_q$.*

If an element $A \in \mathbb{F}_{q^n}$ is nonzero in $\mathbb{F}_q[x]/(f)$ then the canonical representative $g \in \mathbb{F}_q[x]$ of A is invertible modulo f. Then $\gcd(f, g) = 1$ and the Extended Euclidean Algorithm 5.29 computes a representative h of A^{-1} with $O(n^2)$ operations in $\mathbb{F}_q$. In fact, h has degree less than $\deg f = n$. Thus, it is the canonical representative of A^{-1}.

COROLLARY 5.31. *Let the elements of $\mathbb{F}_{q^n}$ be given by a polynomial basis representation. Then the inverse of an element $A \in \mathbb{F}_{q^n}^\times$ can be computed with at most $O(n^2)$ operations in $\mathbb{F}_q$.*

Fast Euclidean Algorithm. Moenck (1973) has described a faster computation of the gcd in Euclidean domains. His algorithm is based on a divide-and-conquer strategy and generalizes a result of Schönhage (1971) for the fast computation of the gcd of two integers. We omit a description of the algorithm and refer to Section 11 in the book of von zur Gathen & Gerhard (1999) instead. Here we only cite the result.

FACT 5.32 (von zur Gathen & Gerhard 1999, Corollary 11.8). *Let $\mathbb{F}_{q^n}$ be represented by a polynomial basis. Then the inverse of an element $A \in \mathbb{F}_{q^n}^{\times}$ can be computed with at most*

$$O(\mathsf{M}(n) \log n) \in O(n \log^2 n \operatorname{loglog} n)$$

operations in $\mathbb{F}_q$.

In Section 8.2, we will compare inversion via the Extended Euclidean Algorithm with an approach which is based on fast exponentiation.

6. Normal bases

A normal bases for $\mathbb{F}_{q^n}$ offers the free evaluation of the Frobenius automorphism. This fact raised interest in (hardware-based) implementations of modern public key cryptosystems; see e.g. Agnew *et al.* (1991), Agnew *et al.* (1993), and IEEE (2000). In this section we collect the main ideas of (matrix-based) arithmetic for *arbitrary normal bases*[39]. In agreement with our model of weighted addition chains with scalar, we give a sketch on multiplication and exponentiation. Moreover, we apply both sequential and parallel exponentiation to the computation of the inverse of an element in $\mathbb{F}_{q^n}$ if it is represented by a normal basis.

The section is organized as follows: After a very short sketch of some basics on normal elements in Section 6.1, we turn to multiplication. We discuss the standard algorithm due to Omura & Massey (1986), and give an example on the construction of the *multiplication matrix* in Section 6.2 for arbitrary normal bases. The subsequent Section 6.3 describes *optimal normal bases* as introduced by Mullin *et al.* (1989). We apply them to exponentiation in Section 6.4. The matrix-based multiplication will appear to be the bottleneck for fast exponentiation in the normal basis representation.

In the closing Section 6.5, we apply exponentiation to the computation of the inverse in $\mathbb{F}_{q^n}$. This subsection mainly presents results that can also be found in von zur Gathen & Nöcker (1999). Our algorithmic approach generalizes an idea of Itoh & Tsujii (1988b). This speeds up the times of sequential inversion as shown by experiment. Moreover, we derive a new parallel algorithm for inversion in $\mathbb{F}_{q^n}$ using the same idea. This parallel algorithm has optimal depth for $q = 2$ and is close to optimal for $q \geq 3$, and the number of processors depends only on q.

6.1. Basics. The *Galois group* $\mathrm{Gal}(\mathbb{F}_{q^n}/\mathbb{F}_q)$ of the extension field of degree n of $\mathbb{F}_q$ is cyclic and the *Frobenius automorphism*

$$\sigma\colon \mathbb{F}_{q^n} \to \mathbb{F}_{q^n} \text{ with } \sigma(A) = A^q$$

is its canonical generator, see Fact 2.9. For an element $A \in \mathbb{F}_{q^n}$ the set of elements $\{\sigma^h(A)\colon 0 \leq h < n\}$ is the set of *conjugates* of A with respect to $\mathbb{F}_q$.

Definition 6.1. *A basis of $\mathbb{F}_{q^n}$ as a vector space of $\mathbb{F}_q$ of the form $\mathcal{N} = (\alpha, \ldots, \alpha^{q^{n-1}})$ consisting of the conjugates of a suitable element $\alpha \in \mathbb{F}_{q^n}$ with respect to $\mathbb{F}_q$, is called a* normal basis *of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$. The element α is then called* normal *(or* free*) in $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$.*

[39] This is in contrast to the subsequent sections where we will study a special type of normal elements.

NORMAL BASIS THEOREM 6.2. *There exists a normal basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$.*

This result was first proven by Hensel (1888). For different types of proofs we refer to the books of Jungnickel (1993), Theorem 3.1.1, and Lidl & Niederreiter (1983), Theorem 2.35.

Normal basis representation of $\mathbb{F}_{q^n}$. Let $\mathcal{N} = (\alpha, \ldots, \alpha^{q^{n-1}})$ be a normal basis of $\mathbb{F}_{q^n}$. Then an element $A = \sum_{0 \le h < n} A_h \alpha^{q^h}$ in $\mathbb{F}_{q^n}$ can be uniquely written as (ordered) linear combination of the basis elements with coefficients $A_0, \ldots, A_{n-1}$ in $\mathbb{F}_q$. We call this the *normal basis representation of* $\mathbb{F}_{q^n}$ with respect to $\mathcal{N}$. Addition of two elements in this basis representation is just coefficient-wise

$$A + B = \sum_{0 \le h < n} A_h \alpha^{q^h} + \sum_{0 \le h < n} B_h \alpha^{q^h} = \sum_{0 \le h < n} (A_h + B_h) \alpha^{q^h}$$

and can be computed with n operations in $\mathbb{F}_q$.

Computing the Frobenius automorphism. Another property of the normal basis representation attracts attention of the experts who work on fast cryptosystems: computing the q-th power of an element is quite easy and fast in both hardware and software since

$$A^q = \sigma(A) = \sigma\Big(\sum_{0 \le h < n} A_h \alpha^{q^h}\Big) = \sum_{0 \le h < n} A_h \sigma(\alpha^{q^h}) = \sum_{0 \le h < n} A_{h-1} \alpha^{q^h}$$

where we set $A_{-1} = A_{n-1}$. Thus, calculating the q-th power is just a cyclic shift of coefficients and can be done without any operations in $\mathbb{F}_q$. Our experiments in Section 6.3 support our assumption that raising to the q-th power is indeed free. For hardware implementations, this arithmetic operation is realized by a *(straightforward) shift register* (Figure 6.1). We observe that the normal basis

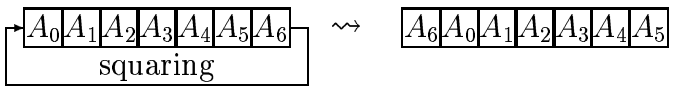


Figure 6.1: Illustration how to square in a normal basis representation of $\mathbb{F}_{2^7}$ over $\mathbb{F}_2$ using a shift register.

representation is a data structure with $c_Q = 0$. Thus, exponentiation can well be parallelized according to Section 4.3.

6.2. The multiplication matrix. The crucial point in (software-based) normal basis arithmetic is multiplication.

Let $C = \sum_{0 \le h < n} C_h \alpha^{q^h}$ be the product of two elements A and B in $\mathbb{F}_{q^n}$ with respect to $\mathcal{N}$. We define the $n \times n$-matrix $T_{\mathcal{N}} = (t_{i,h})_{0 \le i,h < n}$ with entries in $\mathbb{F}_q$ by

$$\alpha^{q^i} \cdot \alpha = \sum_{0 \le h < n} t_{i,h} \alpha^{q^h} \text{ for } 0 \le i < n. \tag{6.3}$$

The matrix is called the *multiplication matrix* of $\mathcal{N}$. A straightforward computation gives for $0 \le j < n$

$$\begin{aligned} \alpha^{q^i} \cdot \alpha^{q^j} &= \left(\alpha^{q^{i-j}} \cdot \alpha\right)^{q^j} \overset{(6.3)}{=} \left(\textstyle\sum_{0 \le h < n} t_{i-j,h} \alpha^{q^h}\right)^{q^j} \\ &= \textstyle\sum_{0 \le h < n} t_{i-j,h} \alpha^{q^{h+j}} = \sum_{0 \le h < n} t_{i-j,h-j} \alpha^{q^h}. \end{aligned} \tag{6.4}$$

Thus, multiplication in the normal basis representation of $\mathbb{F}_{q^n}$ due to $\mathcal{N}$ is completely determined by $T_{\mathcal{N}}$. The coefficients $C_0, \ldots, C_{n-1}$ of the product $C = A \cdot B$ are given due to

$$C_h = \sum_{0 \le i,j < n} A_i \cdot B_j \cdot t_{i-j,h-j} \tag{6.5}$$

by comparison of coefficients:

$$\begin{aligned} \sum_{0 \le h < n} C_h \alpha^{q^h} &= \left(\sum_{0 \le i < n} A_i \alpha^{q^i}\right) \cdot \left(\sum_{0 \le j < n} B_j \alpha^{q^j}\right) = \sum_{0 \le i,j < n} A_i \cdot B_j \cdot \left(\alpha^{q^i} \alpha^{q^j}\right) \\ &\overset{(6.4)}{=} \sum_{0 \le i,j < n} A_i \cdot B_j \cdot \left(\sum_{0 \le h < n} t_{i-j,h-j} \alpha^{q^h}\right) \\ &= \sum_{0 \le h < n} \left(\sum_{0 \le i,j < n} A_i \cdot B_j \cdot t_{i-j,h-j}\right) \alpha^{q^h}. \end{aligned}$$

Obviously, the number of multiplications in $\mathbb{F}_q$ to compute the product of two elements depends on the number $d_{\mathcal{N}}$ of non-zero entries in $T_{\mathcal{N}}$.

FACT 6.6. *Let* $d_{\mathcal{N}} = \#\{(i,h) \in \{0, \ldots, n-1\}^2 : t_{i,h} \neq 0\}$ *be the* density[40] *of* $T_{\mathcal{N}}$. *The product of two elements given by a normal basis representation with respect to* $\mathcal{N}$ *can be computed with at most* $2nd_{\mathcal{N}}$ *multiplications (and at most* $(d_{\mathcal{N}} - 1) \cdot n$ *additions) in* $\mathbb{F}_q$.

[40] Ash *et al.* (1989), p. 192, call this "the complexity of multiplication with respect to the basis" $\mathcal{N}$.

Computation of the multiplication matrix. A common way to compute the multiplication matrix $T_{\mathcal{N}}$ with respect to a normal basis $\mathcal{N} = (\alpha, \ldots, \alpha^{q^{n-1}})$ is to consider the (irreducible) minimal polynomial $f \in \mathbb{F}_q[x]$ of α; see also Jungnickel (1993), Example 3.2.3[41]. Then all products $\alpha^{q^i} \cdot \alpha$ for $0 \le i < n$ are computed in the polynomial basis representation of $\mathbb{F}_{q^n}$ given by the basis $\mathcal{B} = (1, \alpha, \alpha^2, \ldots, \alpha^{n-1})$ with $\alpha = (x \bmod f)$.

We illustrate this by an example which is described in the patent of Omura & Massey (1986), columns 9–10.

EXAMPLE 6.7. (i) Let α be a root of the irreducible polynomial $f = x^7 + x^6 + 1 \in \mathbb{F}_2[x]$. Then

$$\begin{aligned} \alpha &= (x \bmod f) & \alpha^{2^4} &= (x^6 + x^5 + x^4 + x^3 + x \bmod f) \\ \alpha^2 &= (x^2 \bmod f) & \alpha^{2^5} &= (x^5 + x^4 + x^2 + x + 1 \bmod f) \\ \alpha^{2^2} &= (x^4 \bmod f) & \alpha^{2^6} &= (x^4 + x^3 + 1 \bmod f) \\ \alpha^{2^3} &= (x^6 + x + 1 \bmod f). \end{aligned}$$

Hence,

$$\begin{aligned} & \sum_{0 \le h < 7} A_h \alpha^{q^h} \\ = \; & (A_3 + A_4)x^6 + (A_4 + A_5)x^5 + (A_2 + A_4 + A_5 + A_6)x^4 \\ & + (A_4 + A_6)x^3 + (A_1 + A_5)x^2 + (A_0 + A_3 + A_4 + A_5)x \\ & + (A_3 + A_5 + A_6) \cdot 1 \bmod f. \end{aligned}$$

To check that f is irreducible, we have to check (with Gaussian elimination) that the right-hand side is zero if and only if $A_0 = \cdots = A_6 = 0$, i.e. the roots of f are indeed linearly independent.

(ii) The computation of the products $\alpha^{q^i} \cdot \alpha$ for $0 \le i < 7$ in $\mathbb{F}_q[x]/(f)$ gives

$$\begin{aligned} \alpha \cdot \alpha &= (x^2 \bmod f) & \alpha^{2^4} \cdot \alpha &= (x^5 + x^4 + x^2 + 1 \bmod f) \\ \alpha^2 \cdot \alpha &= (x^3 \bmod f) & \alpha^{2^5} \cdot \alpha &= (x^6 + x^5 + x^3 + x^2 + x \bmod f) \\ \alpha^{2^2} \cdot \alpha &= (x^5 \bmod f) & \alpha^{2^6} \cdot \alpha &= (x^5 + x^4 + x \bmod f) \\ \alpha^{2^3} \cdot \alpha &= (x^6 + x^2 + x + 1 \bmod f). \end{aligned}$$

[41] One way to find a normal element and its minimal polynomial at the same time is as follows. One starts with an irreducible polynomial of degree n and checks whether its roots are linearly independent. If an irreducible polynomial f with linearly independent roots is found then the root α of f is normal in $\mathbb{F}_{q^n}$ and f is its minimal polynomial over $\mathbb{F}_q$.

(iii) We can perform Gaussian elimination on the following matrix to write these 7 products as linear combinations of the conjugates of α. For $\alpha^{2^i} = \sum_{0\le j<7} A_{j,i}x^j \bmod f$ the entry $(j+1, i+1)$ in the left part of the matrix is $A_{j,i}$. The column $8+i$ of the right part contains the coefficients of the polynomial representation of $\alpha^{2^i}\cdot\alpha$.

$$\left(\begin{array}{ccccccc|ccccccc}
0&0&0&1&0&1&1&0&0&0&1&1&0&0\\
1&0&0&1&1&1&0&0&0&0&1&0&1&1\\
0&1&0&0&0&1&0&1&0&0&1&1&1&0\\
0&0&0&0&1&0&1&0&1&0&0&0&1&0\\
0&0&1&0&1&1&1&0&0&0&0&1&0&1\\
0&0&0&0&1&1&0&0&0&1&0&1&1&1\\
0&0&0&1&1&0&0&0&0&0&1&0&1&0
\end{array}\right)$$

The output is then $(E_7|T')$ with E_7 the 7×7-identity matrix and

$$T' = \begin{pmatrix}
0&1&0&0&1&0&1\\
1&1&0&1&0&1&0\\
0&0&0&0&0&1&1\\
0&1&1&1&0&0&1\\
0&1&1&0&0&1&1\\
0&1&0&0&1&0&0\\
0&0&1&0&0&0&1
\end{pmatrix}$$

(iv) The multiplication matrix $T_{\mathcal{N}}$ is the transposed matrix of T'. Entry (j,i) in T' contains $B_{j-1,i-1}$ such that $\alpha^{2^i}\cdot\alpha = \sum_{0\le j<7} B_{j,i}\alpha^{2^j}$. Then

$$\begin{aligned}
\alpha\cdot\alpha &= \alpha^2 & \alpha^{2^4}\cdot\alpha &= \alpha^{2^5}+\alpha\\
\alpha^2\cdot\alpha &= \alpha^{2^5}+\alpha^{2^4}+\alpha^{2^3}+\alpha^2+\alpha & \alpha^{2^5}\cdot\alpha &= \alpha^{2^4}+\alpha^{2^2}+\alpha^2\\
\alpha^{2^2}\cdot\alpha &= \alpha^{2^6}+\alpha^{2^4}+\alpha^{2^3} & \alpha^{2^6}\cdot\alpha &= \alpha^{2^6}+\alpha^{2^4}+\alpha^{2^3}+\alpha^{2^2}+\alpha\\
\alpha^{2^3}\cdot\alpha &= \alpha^2+\alpha^{2^3}.
\end{aligned}$$

Usually, the multiplication matrix $T_{\mathcal{N}}$ is computed once and then stored. All other multiplications of elements of $\mathbb{F}_{q^n}$—given by a normal basis representation due to $\mathcal{N} = (\alpha,\dots,\alpha^{q^{n-1}})$—can be done without knowing the minimal polynomial f of α. ◊

REMARK 6.8. *Let $\mathcal{N} = (\alpha,\dots,\alpha^{q^{n-1}})$ be a normal basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$, and let $f\in\mathbb{F}_q[x]$ be the minimal polynomial of α. The multiplication matrix $T_{\mathcal{N}}$ can be computed with at most $n-1$ many q-th powers (to compute α^{q^h} for $1\le h<n$)*

and n multiplications (to compute $\alpha^{q^i} \cdot \alpha$ for $0 \leq i < n$) in $\mathbb{F}_q[x]/(f)$ and a Gaussian elimination[42] *on a $n \times 2n$-matrix over $\mathbb{F}_q$. This yields a total of at most*

$$\begin{aligned} & (3\mathsf{M}(n)+n)\ell_2(q) \cdot (n-1) + (3\mathsf{M}(n)+n) \cdot n + O(n^3) \\ \in\ & O(n^3 + n^2 \log n \cdot \operatorname{loglog} n \cdot \log q) \end{aligned}$$

operations in $\mathbb{F}_q$ using the results on a polynomial basis representation with arbitrary modulus as in Corollary 5.16.

Hardware-based implementation. Omura & Massey (1986) hold a patent on a hardware-based implementation of the matrix-based multiplication algorithm described above. Wang *et al.* (1985) report on a chip compiled according to this patent and some improvements. Onyszchuk *et al.* (1988) have got another patent on hardware-based normal basis multiplication. The architecture and the algorithms of the latter one are described on its cryptographic background by Rosati (1989) and Agnew *et al.* (1991), respectively. See Geiselmann (1994), Section 3.1.1, for a comparative discussion on both implementations.

The so-called *Massey-Omura multiplier* hardwires a multiplication matrix $\bar{T}_{\mathcal{N}} = (\bar{t}_{i,h})_{0 \leq i,h < n}$. The entries of $\bar{T}_{\mathcal{N}}$ are given by a permutation of the entries of the multiplication matrix $T_{\mathcal{N}}$ such that $\bar{t}_{i,h} = t_{i-h,-h}$ for $0 \leq i, h < n$. Then the coefficients $C_0, \ldots, C_{n-1}$ of the product $C = A \cdot B$ are given by

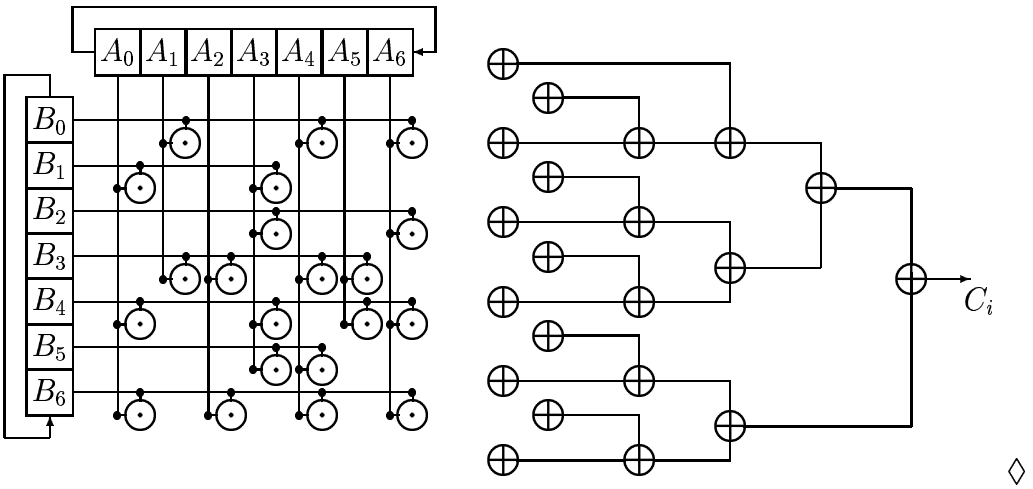
$$\begin{aligned} \sum_{0 \leq h < n} C_h \alpha^{q^h} &= \left(\sum_{0 \leq i < n} A_i \alpha^{q^i} \right) \left(\sum_{0 \leq j < n} B_j \alpha^{q^j} \right) \\ &= \sum_{0 \leq h < n} \left(\sum_{0 \leq i,j < n} A_i \cdot B_j \cdot t_{i-j,h-j} \right) \alpha^{q^h} = \sum_{0 \leq h < n} \left(\sum_{0 \leq i,j < n} A_{i-h} \cdot \bar{t}_{i,j} \cdot B_{j-h} \right) \alpha^{q^h}. \end{aligned}$$

The matrix can be hardwired. Only the coefficients $A_0, \ldots, A_{n-1}$ and $B_0, \ldots, B_{n-1}$ have to be shifted with respect to h. This can easily be done by two shift registers. The corresponding algorithm is also suggested in IEEE (2000), A.3.8.

EXAMPLE 6.7 CONTINUED. The schematic Massey-Omura multiplier for $\mathcal{N} = (\alpha, \ldots, \alpha^{q^{n-1}})$ with α a root of $f = x^7 + x^6 + 1 \in \mathbb{F}_2[x]$ is described in the following figures:[43]

[42] A Gaussian elimination on a $n \times 2n$-matrix with entries in $\mathbb{F}_q$ can be performed with $O(n^3)$ operations in $\mathbb{F}_q$, see von zur Gathen & Gerhard (1999), Section 25.5

[43] We omit the connection between the AND-gates and the XOR-gates. Two of each AND-gates are combined with one XOR-gate.



◊

Following Wang *et al.* (1985) and Beth *et al.* (1991), a Massey-Omura multiplier for a normal basis $\mathcal{N}$ in $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$ can be realized with $2n$ registers ($\square$), $d_{\mathcal{N}}$ AND gates ($\odot$) and $d_{\mathcal{N}} - 1$ XOR gates ($\oplus$).

6.3. Optimal normal bases. By Fact 6.6, the density $d_{\mathcal{N}}$ of a normal basis is the crucial parameter for both software- and hardware-implemented multiplication. Beth *et al.* (1991) report on empirical examinations[44] which showed an average value for $d_{\mathcal{N}}$ of $\frac{q-1}{q} \cdot n^2$. This is quadratic in the degree n.

A lower bound. In this paragraph, we follow Mullin *et al.* (1989). They have proven a sharp lower bound on $d_{\mathcal{N}}$ which is only linear in n.

A useful tool is the *trace*. The trace map $\mathrm{Tr}_{\mathbb{F}_{q^n}/\mathbb{F}_q} : \mathbb{F}_{q^n} \to \mathbb{F}_{q^n}$ is defined by

$$\mathrm{Tr}_{\mathbb{F}_{q^n}/\mathbb{F}_q}(A) = \sum_{0 \le i < n} A^{q^i} \tag{6.9}$$

for each $A \in \mathbb{F}_{q^n}$. We often write $\mathrm{Tr}_{q^n/q}$ short for $\mathrm{Tr}_{\mathbb{F}_{q^n}/\mathbb{F}_q}$. For all $A \in \mathbb{F}_{q^n}$, we have

$$\begin{aligned} \mathrm{Tr}_{q^n/q}(A)^q &= \left(\sum_{0 \le i < n} A^{q^i} \right)^q = \sum_{0 \le i < n} A^{q^{i+1}} = \sum_{1 \le i < n} A^{q^i} + A^{q^n} \\ &= A + \sum_{1 \le i < n} A^{q^i} = \sum_{0 \le i < n} A^{q^i} = \mathrm{Tr}_{q^n/q}(A) \end{aligned}$$

using $A^{q^n} = A$ in $\mathbb{F}_{q^n}^\times$ according to Fermat's Little Theorem 2.3. Thus, the element $\mathrm{Tr}_{q^n/q}(A)$ is an element in $\mathbb{F}_q$ for all $A \in \mathbb{F}_{q^n}$ by (2.8).

[44] "An average value of $\frac{q-1}{q} \cdot n^2$ for C_N is found from empirical examinations. This is the obvious result of modeling with random matrices as well.", Beth *et al.* (1991), p. 174 [the C_N is our $d_{\mathcal{N}}$].

FACT 6.10 (Mullin *et al.* 1989, Theorem 2.1). *Let $\mathcal{N}$ be a normal basis in $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ with density $d_{\mathcal{N}}$. Then $d_{\mathcal{N}} \geq 2n-1$.*

PROOF. The proof is taken from Menezes *et al.* (1993): Let α be a normal element in $\mathbb{F}_{q^n}$ and $\mathcal{N} = (\alpha, \ldots, \alpha^{q^{n-1}})$ the corresponding basis over $\mathbb{F}_q$. The trace $A = \mathrm{Tr}_{q^n/q}(\alpha) = \sum_{0 \leq i < n} \alpha^{q^i}$ of α over $\mathbb{F}_q$ is an element in $\mathbb{F}_q$. Let $T_{\mathcal{N}} = (t_{i,h})_{0 \leq i,h < n}$ be the multiplication matrix of $\mathcal{N}$. Then

$$\begin{aligned} A \cdot \alpha &= \mathrm{Tr}_{q^n/q}(\alpha) \cdot \alpha = \sum_{0 \leq i < n} (\alpha^{q^i} \cdot \alpha) \\ &= \sum_{0 \leq i < n} \sum_{0 \leq h < n} t_{i,h} \alpha^{q^h} = \sum_{0 \leq h < n} \left(\sum_{0 \leq i < n} t_{i,h} \right) \alpha^{q^h}, \end{aligned}$$

and comparison of coefficients proves

$$\sum_{0 \leq i < n} t_{i,h} = \begin{cases} A & \text{if } h = 0, \\ 0 & \text{else.} \end{cases} \tag{6.11}$$

Because $\mathcal{N}' = (\alpha^{q^i} \cdot \alpha)_{0 \leq i < n}$ is also a normal basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$, the matrix $T_{\mathcal{N}}$ is invertible. Hence, there has to be at least one non-zero element in each row of $T_{\mathcal{N}}$. With respect to (6.11), there are at least two non-zero elements in the i-th row for $1 \leq i < n$. In row $i = 0$, there is at least one non-zero entry. Summing up, we have $d_{\mathcal{N}} \geq 2(n-1)+1$. □

Following Mullin *et al.* (1989), we call a normal basis $\mathcal{N}$ *optimal* when $d_{\mathcal{N}} = 2n-1$. This lower bound is sharp! Mullin *et al.* (1989) constructed two different sets of normal bases with optimal density. In there early paper, they restrict their construction to $\mathbb{F}_{2^n}$. Fact 6.13 below is proven in Menezes *et al.* (1993). It gives a construction of optimal normal bases with the help of primitive roots of unity in $\mathbb{F}_{q^n}$.

DEFINITION 6.12. *Let r be a positive integers and q be a prime power such that $\gcd(r,q) = 1$. An element ζ in an extension field of $\mathbb{F}_q$ is called a* primitive r-th root of unity *if $\zeta^r = 1$ and $\zeta^s \neq 1$ for $0 < s < r$.*

Since $\zeta^r - 1 = 0$, we have $x^r - 1 \in \mathbb{F}_q[x]$ a multiple of the minimal polynomial $\mu \in \mathbb{F}_q[x]$ of ζ, i.e. $\zeta \in \mathbb{F}_{q^r}$. We recall some notation: We write $\langle a_1, \ldots, a_k \rangle \subseteq \mathbb{Z}_r^\times$ for the smallest subgroup of $\mathbb{Z}_r^\times$ containing all elements $a_1, \ldots, a_k$.

FACT 6.13 (Menezes *et al.* 1993, Theorems 5.2 and 5.3). *Let $n \in \mathbb{N}_{\geq 1}$.*

(i) Let $n+1$ be prime and q be a prime power generating $\mathbb{Z}_{nk+1}^\times$, i.e. $\langle q \rangle = \mathbb{Z}_{n+1}^\times$. Let ζ be a primitive $(n+1)$-th root of unity in an extension field of $\mathbb{F}_q$ and $\alpha = \zeta$. Then $\mathcal{N} = (\alpha, \ldots, \alpha^{q^{n-1}})$ is an optimal normal basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$.

(ii) Let $2n+1$ be a prime. If either $\langle 2\rangle = \mathbb{Z}_{2n+1}^{\times}$ or $\langle 2, \{1,-1\}\rangle = \mathbb{Z}_{2n+1}^{\times}$. Then $\alpha = \zeta + \zeta^{-1}$ generates an optimal normal basis $\mathcal{N} = (\alpha, \ldots, \alpha^{q^{n-1}})$ of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$, where ζ is a primitive $(2n+1)$-th root of unity.

These constructions describe all optimal normal bases! Gao & Lenstra (1992) proved this somehow surprising result. Unfortunately, this is bad news for fast arithmetic in $\mathbb{F}_{q^n}$. Optimal normal bases do not exist for all finite fields $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$. The bold figures in Table 6.1 show the percentage of all extension fields over small prime fields of degree less than 10000 for which an optimal normal basis exist.

Normal bases with low density. Ash *et al.* (1989) have generalized the idea of Fact 6.13 to construct what they call *low complexity normal bases* for $\mathbb{F}_{2^n}$; these normal bases are called *Gaussian normal bases* in IEEE (2000). A further generalization for $\mathbb{F}_{q^n}$ has been given by Wassermann (1990), Satz 1, and Beth *et al.* (1991), Theorem 5.

FACT 6.14 (Beth *et al.* 1991, Theorem 5). *Let k and n be positive integers, and q a prime power such that $nk+1$ is a prime not dividing q, and let $\mathcal{K}$ be the uniquely determined subgroup of order k in $\mathbb{Z}_{nk+1}^{\times}$. If $\langle q, \mathcal{K}\rangle = \mathbb{Z}_{nk+1}^{\times}$ and if ζ is a primitive $(nk+1)$-th root of unity in $\mathbb{F}_{q^{nk}}$. Then $\alpha = \sum_{a\in\mathcal{K}} \zeta^a$ is normal in $\mathbb{F}_{q^n}$ and $\mathcal{N} = (\alpha, \ldots, \alpha^{q^{n-1}})$ is a normal basis in $\mathbb{F}_{q^n}$.*

FACT 6.15 (Beth *et al.* 1991, Theorem 6). *The density of a normal basis $\mathcal{N}$ as in Fact 6.14 is bounded by $d_{\mathcal{N}} \le n(k+1) - k$.*

This construction cannot be done for all finite fields. We give some empirical values on the distribution of those normal bases in Table 6.1. The theoretical result was proven by Wassermann (1993).

FACT 6.16 (Wassermann 1993, Satz 3.3.4). *Let p be a prime, e and n be positive integers, and $q = p^e$. There is a $k \in \mathbb{N}_{\ge 1}$ such that a normal basis can be constructed as in Fact 6.14 if and only if q and n satisfy*

$$\gcd(e,n) = 1 \text{ and } \begin{array}{ll} 2p \not| n, & \text{if } p \equiv 1 \bmod 4, \\ 4p \not| n, & \text{if } p \equiv 2,3 \bmod 4. \end{array}$$

Experiments. We recall the 50 finite fields $\mathbb{F}_{2^n}$ which are subject of the three test series for polynomial basis representation. Each of these fields has an optimal normal basis over $\mathbb{F}_2$. We fixed the corresponding normal basis with parameters (n,k) over $\mathbb{F}_2$, where $k \in \{1,2\}$, and collected them in a fourth test series *Normal*.

We computed the multiplication matrix for each normal basis using an algorithm of Wassermann (1990). We will describe a generalized version of this

Existence of normal bases constructed due to Fact 6.14 with given parameter $k \in \mathbb{N}_{\geq 1}$								
$k \quad \backslash \quad q$	2	3	5	7	11	13	17	19
$k = 1$	4.70	**4.76**	**4.92**	**4.65**	**4.43**	**4.57**	**4.50**	**4.72**
$k \leq 2$	**17.07**	18.40	17.62	17.37	17.14	17.11	16.89	17.26
$k \leq \log_2 n$	63.34	66.63	65.45	69.66	69.89	69.63	69.78	69.77
$k \leq \sqrt{n}$	86.86	91.02	89.34	95.75	96.98	95.32	96.19	97.77
$k < \infty$	87.51	91.67	90.01	96.43	97.73	96.16	97.06	98.69
theo.	87.51	91.67	90.01	96.43	97.73	96.16	97.06	98.69

Table 6.1: Percentage of field extensions $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ with $2 \leq n < 10000$ for which there is a normal basis as constructed in Fact 6.14 over $\mathbb{F}_q$. The parameter q is given in the top row; e.g. the figures for $\mathbb{F}_2$ are in the second column. The rows show the distribution if the value $k \in \mathbb{N}_{\geq 1}$ is also limited. We have confined our experiments for $r = nk + 1$ by $r < 1000000$. Bold figures are the percentage of optimal normal bases for given q. The last row labeled *theo.* quantifies the result of Fact 6.16.

algorithm in Section 8.3. The multiplication matrix is sparse for optimal normal bases. We have stored a non-zero entry $t_{i,h}$ of this sparse matrix by its indices. An element $A \in \mathbb{F}_{2^n}$ was represented by an array of bits collected in machine words of 32 bits each. Each bit is a coefficient A_i in the linear combination $\sum_{0 \leq i < n} A_i \alpha^{2^i}$ of A. Our experiment consisted of 1000 trials for multiplication as well as for squaring in $\mathbb{F}_{2^n}$ given by the normal basis $\mathcal{N} = (\alpha, \alpha^2, \ldots, \alpha^{2^{n-1}})$. For each trial we chose elements of $\mathbb{F}_{2^n}$ at random. Squaring was realized by a cyclic shift on the bits which is supported for words by the processor. The times are documented in Table A.12; they are illustrated in Figure 6.2. These times support the assumption of the theoretical estimates that squaring is free. For multiplication, our experiments show the predicted quadratic running time with respect to the degree of the field extension n. But the comparison with polynomial multiplication is worse than expected. Also the running times for $k = 1$ and $k = 2$ differ by a factor of nearly $\frac{5}{3}$ which contradicts with Fact 6.6 since we have $d_{\mathcal{N}} = 2n - 1$ for both cases. The problem occurs, because the implementation of the multiplication routine is software-based but bit-oriented. Each operation beneath the word level of a processor is expensive. And the sparseness of the multiplication matrix forces the program to jump between bit locations. This is expensive. The difference between $k = 1$ and $k = 2$ illustrates this. For $k = 1$, the matrix $T_{\mathcal{N}}$ has one row for which each entry is non-zero. All other rows have exactly one non-zero entry. We have implemented a word-level oriented subroutine for this particular row. For $k = 2$, all rows have exactly 2 non-zero entries. Thus, an analogous shortcut cannot be done. Our experiment show that software-based

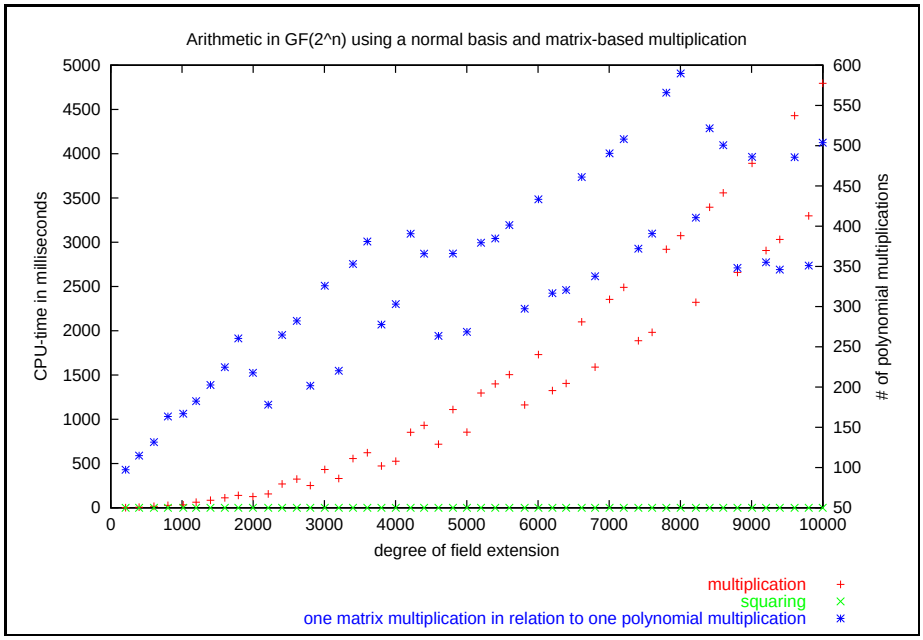


Figure 6.2: Times for multiplication (+) and squaring (×) for the test series *Normal*. The times are the average of 1000 trials. Here $\mathbb{F}_{2^n}$ is given by a normal basis representation with respect to an optimal normal basis $\mathcal{N} = (\alpha, \ldots, \alpha^{2^{n-1}})$. The times are marked on the left y-axis. The scale of the right y-axis relates to the quotient (∗) between the time for one matrix-based multiplication in the chosen normal basis, and the time for one polynomial multiplication of two polynomials of degree at most $n-1$ over $\mathbb{F}_2$.

arithmetic for a normal basis representation using the multiplication matrix is too slow.

6.4. Exponentiation in a normal basis representation. As for polynomial bases, we summarize the facts focusing on parallel exponentiation. Here multiplication is matrix-based multiplication. The significance of optimal normal bases in the above references makes us to distinguish between arbitrary, low density, and optimal normal bases. As for polynomials, we restrict ourselves to finite fields $\mathbb{F}_{2^n}$. While arbitrary normal bases exist for all finite fields, the two other types of normal elements do not always exist, as documented in Table 6.1. As remarked above, the multiplication is slow compared to polynomial multiplication. If we use an optimal instead of an arbitrary normal basis then the bound drops from cubic to quadratic. Nevertheless, this is slower than exponentiation using a sparse irreducible modulus f and Karatsuba's algorithm, see Tables 5.5 and 5.6. The experiments underline this disadvantage of matrix-based multiplication for a normal basis representation.

Table 6.3 shows the high possible speed-up when computing exponentiation

Normal basis representation: multiplication matrix			
	arbitrary	low density	optimal
arithmetic			
mult. (c_A)	$\leq 2n^3$	$2(k+1)n^2 - 2kn$	$4n^2 - 2n$
squaring (c_Q)	0	0	0
$c = \frac{c_Q}{c_A}$	0	0	0
exponentiation			
sequential	$2\frac{n^4}{\log_2 n}(1 + o(1))$	$2(k+1)\frac{n^3}{\log_2 n}(1 + o(1))$	$4\frac{n^3}{\log_2 n}(1 + o(1))$
parallel	$\leq 2n^3\lceil \log_2 n \rceil$	$\leq 2(k+1)n^2\lceil \log_2 n \rceil$	$\leq 4n^2\lceil \log_2 n \rceil$

Table 6.3: Bounds on exponentiation in $\mathbb{F}_{2^n}$ given by a normal basis representation with matrix-based multiplication.

in parallel. This is caused by free squarings. Parallel computing decreases the total running time for exponentiation by nearly one order of magnitude compared to sequential exponentiation. Nevertheless, the best bound on parallel exponentiation is only $O(n^2 \log n)$. Table 5.5 shows the same asymptotic estimates for a polynomial basis representation of $\mathbb{F}_{2^n}$ with sparse modulus and classical multiplication. The fastest polynomial multiplication algorithm gives a bound of only $O(n^2)$ for this data structure. Thus, a normal basis representation is beaten by a special polynomial basis representation. The reason for this is that matrix-based multiplication is too slow. Therefore, we discuss in the subsequent Sections 7–9 how to substitute these multiplication algorithm by faster ones for a special type of normal bases, namely the Gauß periods.

6.5. Division in a normal basis representation. In a normal basis representation of $\mathbb{F}_{q^n}$ division by an element $B \in \mathbb{F}_{q^n}^\times$ is usually computed by multiplication with the *inverse* $B^{-1} \in \mathbb{F}_{q^n}^\times$. The inverse can be calculated by exponentiation since

$$B^{q^n-2} = B^{q^{n-1}} \cdot B^{-1} = 1 \cdot B^{-1} = B^{-1}$$

by Fermat's Little Theorem 2.3. Setting $e = \frac{q^{n-1}-1}{q-1}$ for $n \in \mathbb{N}_{\geq 1}$ and $q \geq 2$, the exponent $e' = q^n - 2$ is just

$$e' = q^n - 2 = \frac{q^{n-1}-1}{q-1} \cdot (q-1)q + (q-2) = e \cdot (q-1)q + (q-2). \tag{6.17}$$

For $q = 2$, this collapses to $e' = 2^n - 2 = 2 \cdot (2^{n-1} - 1)$ with $e = 2^{n-1} - 1$. The exponent e has the q-ary representation

$$(e)_q = (\underbrace{1, \ldots, 1}_{n-1})$$

which is composed of $n-1$ many ones. An exponent of this special type is called a *repunit*, see Beiler (1966)[45]. This observation is our key to prove the following bound on division.

RESULT 6.18. *Let $\mathcal{N}$ be a normal basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$, and suppose that the elements are given in a normal basis representation due to $\mathcal{N}$. Then division by an element can be computed with at most*

$$2nd_{\mathcal{N}} \cdot ((\log_2(n-1) + \log_2(q-1)) \cdot (1 + o(1)) + 3)$$

operations in $\mathbb{F}_q$. If $\mathcal{N}$ is optimal then division can be computed with at most

$$(4n^2 - 2n) \cdot ((\log_2(n-1) + \log_2(q-1)) \cdot (1 + o(1)) + 3)$$

operations in $\mathbb{F}_q$.

We mainly follow von zur Gathen & Nöcker (1999) To prove this result. Our algorithm works for all $q \in \mathbb{N}_{\geq 2}$. In the special binary case, i.e. $q = 2$, this general approach covers results of Wang *et al.* (1985), Itoh & Tsujii (1988b), Asano *et al.* (1989), and Xu (1990) as we will show in Example 6.25.

We solve the exponentiation problem B^{q^n-2} by computing a q-addition chain for the exponent $e' = q^n - 2$. The first step is a reduction to *repunits* which are integers with q-ary representation $(1, \ldots, 1)$.

LEMMA 6.19. *Let q and n be positive integers, q greater than 1, and ε be a q-addition chain with weight (c_Q, c_A) computing $e = \frac{q^{n-1}-1}{q-1}$. Furthermore, whenever $q > 2$, we choose ψ a q-addition chain of weight (c_Q, c_A) for $q-2$. Then there is a q-addition chain γ of weight (c_Q, c_A) for $e' = q^n - 2$ of (sequential) length at most*

$$L(\gamma) = \begin{cases} c_A \cdot (A(\varepsilon) + A(\psi) + 2) + c_Q \cdot (Q(\varepsilon) + Q(\psi) + 1) & \text{if } q > 2, \text{ and} \\ c_A \cdot A(\varepsilon) + c_Q \cdot (Q(\varepsilon) + 1) & \text{if } q = 2. \end{cases}$$

PROOF. We have $q^n - 2 = e \cdot (q-1) \cdot q + (q-2)$ by (6.17). If $q = 2$ then $2^n - 2 = 2 \cdot e$ and γ is just ε supplemented by a last doubling $(e, -q)$ as claimed by the formula. For $q > 2$, let ψ' be a q-addition chain with $\mathcal{S}(\psi') = \mathcal{S}(\psi) \cup \{q-1\}$ and $\mathcal{R}(\psi') = \mathcal{R}(\psi) \cup \{(q-2, 1)\}$. Construct the q-addition chain $\gamma \leftarrow \psi' \odot \varepsilon$ by concatenation. It computes $(q-1) \cdot e$ with $A(\psi') + A(\varepsilon)$ non-q-steps and $Q(\psi') + Q(\varepsilon)$ many q-steps. Adding a q-step includes the node $e(q-1) \cdot q$ in this chain. Finally, the non-q-step $(e \cdot (q-1) \cdot q, q-2)$ is added to the chain γ for e'. Substituting $A(\psi')$ by $A(\psi) + 1$ and $Q(\psi')$ by $Q(\psi)$ completes the proof. □

[45] "[...] and for convenience the author has used the term 'repunit number' (repeated unit) to represent mono-digit numbers consisting solely of the digit 1.", Beiler (1966), p. 83.

Addition chains with scalar for repunits. Of course, one may compute a q-addition chain for a repunit $e = \frac{q^m-1}{q-1}$ with $q, m \in \mathbb{N}_{\geq 2}$ by the algorithms given in Section 3. The resulting addition chain has at least $m-1$ many q-steps and $\frac{m}{\log m}(1+o(1))$ non-q-steps. But this can be improved by exploiting the special structure of $(e)_q$. The following equation is our key to get a short chain for e:

$$\begin{array}{l} \underbrace{(1,\ldots,1,1,\ldots,1)}_{m_2+m_1} = \underbrace{(1,\ldots,1}_{m_2},\underbrace{0,\ldots,0)}_{m_1} + \underbrace{(1,\ldots,1)}_{m_1} \\ \sum_{0\leq i<(m_2+m_1)} q^i \;\; = \left(\sum_{0\leq i<m_2} q^i\right)\cdot q^{m_1} + \sum_{0\leq i<m_1} q^i \end{array} \tag{6.20}$$

This equation points a way to compose two q-addition chains for the repunits $e_2 = \frac{q^{m_2}-1}{q-1}$ and $e_1 = \frac{q^{m_1}-1}{q-1}$ to get a chain for the repunit $\frac{q^{m_1+m_2}-1}{q-1}$. The observation that this equation only depends on m_1 and m_2 reduces the problem to find a q-addition chain for $e = \frac{q^m-1}{q-1}$ to that of getting an (original) addition chain for $m \in \mathbb{N}_{\geq 1}$.

ALGORITHM 6.21. q-addition chain for repunits.

Input: Integers $q \geq 2$ and $m \geq 2$, and an (original) addition chain μ for m with $\mathcal{S}(\mu) = \{a_0, \ldots, a_L\}$ and $\mathcal{R}(\mu) = \{(a_{j(1)}, a_{k(1)}), \ldots, (a_{j(L)}, a_{k(L)})\}$.
Output: A q-addition chain ε computing $e = \frac{q^m-1}{q-1}$.

1. Set ε the empty addition chain with $\mathcal{S}(\varepsilon) = \{1\}$.
2. For $1 \leq i \leq L$ do 3–5
3. For $1 \leq h \leq a_{k(i)}$ do
4. If $\frac{q^{a_{j(i)}+h}-q^h}{q-1}$ is not in $\mathcal{S}(\varepsilon)$ then add the node $\frac{q^{a_{j(i)}+h}-q^h}{q-1}$ and the rule $(\frac{q^{a_{j(i)}}-1}{q-1}\cdot q^{h-1}, -q)$ to ε.
5. Add the node $\frac{q^{a_{j(i)}+a_{k(i)}}-1}{q-1}$ and the rule $(\frac{q^{a_{j(i)}}-1}{q-1}\cdot q^{a_{k(i)}}, \frac{q^{a_{k(i)}}-1}{q-1})$ to ε.
6. Return ε.

LEMMA 6.22. *Algorithm 6.21 computes a q-addition chain for $e = \frac{q^m-1}{q-1}$ with $A(\varepsilon) = L(\mu)$ non-q-steps and $Q(\varepsilon) \leq \sum_{0\leq i\leq L(\mu)} a_{k(i)}$ many q-steps.*

PROOF. We prove correctness by induction on i. For $i = 0$, we have $a_0 = 1$ and $\frac{q^1-1}{q-1} = 1$. This node is in the empty addition chain that is initialized in step 1. Thus, we suppose that all nodes $\frac{q^{a_{i'}}-1}{q-1}$ for $0 \leq i' < i$ are in ε before lap i. Then the node $\frac{q^{a_{j(i)}+a_{k(i)}}-1}{q-1}$ is computed in steps 3–5 according to (6.20). Because we have $a_i = a_{j(i)} + a_{k(i)}$, correctness follows. For each $1 \leq i \leq L$, a single non-q-step is added to ε in steps 3–4. In lap $1 \leq i \leq L$ at most $a_{k(i)}$ many q-steps are added. We sum up to get the claimed number of steps. □

For (original) star addition chains with $q = 2$ the above algorithm was already described by Brauer (1939). For a star addition chain μ we have $j(i) = i - 1$, and therefore,

$$\sum_{1 \le i \le L(\mu)} a_{k(i)} = \sum_{1 \le i \le L(\mu)} (a_i - a_{j(i)}) = \sum_{1 \le i \le L(\mu)} (a_i - a_{i-1}) \\ = a_{L(\mu)} - a_0 = a_{L(\mu)} - 1.$$

COROLLARY 6.23. *Let q be an integer greater than 1, and μ be a star addition chain for a positive integer m. Then there is a q-addition chain ε for $e = \frac{q^m-1}{q-1}$ with $A(\varepsilon) = L(\mu)$ non-q-steps and $Q(\varepsilon) = m - 1$ many q-steps.*

In the view of Lemma 6.19, this gives the following result.

THEOREM 6.24. *Let B be an element in $\mathbb{F}_{q^n}^\times$. The inverse B^{-1} can be computed with at most*

$$\begin{array}{ll} c_A \cdot (\ell_2^\star(n-1) + \ell_2(q-2) + 2) + c_Q \cdot (n-1) & \text{if } q > 2, \text{ and} \\ c_A \cdot \ell_2^\star(n-1) + c_Q \cdot (n-1) & \text{if } q = 2. \end{array}$$

operations in $\mathbb{F}_q$.

A normal basis $\mathcal{N}$ has weight $(c_Q, c_A) = (0, 2nd_\mathcal{N})$ by Fact 6.6, and $d_\mathcal{N} = 2n - 1$ when $\mathcal{N}$ is optimal. Furthermore, $\ell_2^\star(n-1) \le \log_2(n-1) + \frac{\log_2(n-1)}{\log_2 \log_2(n-1)}(1 + o(1))$ by Algorithm 3.12, and $\ell_2(q-2) \le \log_2(q-2) + \frac{\log_2(q-2)}{\log_2 \log_2(q-2)}(1 + o(1))$ for $q > 2$ by (3.19). We substitute the division by multiplication by the inverse. Hence, we have to add one further multiplication to the operations above. Then with Theorem 6.24, these bounds prove Result 6.18.

EXAMPLE 6.25. We focus on $q = 2$. Let B be an element in $\mathbb{F}_{2^n}^\times$ given in a normal basis representation. By choosing specific addition chains μ for $m = n - 1$ as input of Algorithm 6.21, we get the following versions that were already discussed in the literature.

- Wang *et al.* (1985), Section IV, suggested to choose the *linear addition chain* μ with $\mathcal{S}(\mu) = \{1, 2, \ldots, n-1\}$.[46] Then the inverse of B can be computed with $n - 1$ multiplications in $\mathbb{F}_{2^n}$.

- Xu (1990) described inversion in $\mathbb{F}_{2^n}$ using the following addition chain[47] γ for $n - 1$: Choose a parameter $m' \in \{1, \ldots n-1\}$. First, compute the

[46] Indeed, they integrated the final squaring and compute $2 \cdot \sum_{0 \le h < i} 2^i$ as the i-th element of their chain.

[47] Xu (1990) also computed the addition chain for $2^n - 2$, integrating the final squaring.

linear addition chain for m' with $\mathcal{S}(\gamma) = \{1, 2, \ldots, m'\}$, and set $m_1 = n-1-(\lceil \frac{n-1}{m'} \rceil - 1) \cdot m' \leq m'$. In a second step add the nodes $m_1 + m' \cdot j$ and the rules $(m_1 + m'(j-1), m')$ for $1 \leq j < \lceil \frac{n-1}{m'} \rceil$ to γ. For $m' \approx \sqrt{\frac{1}{3}(n-1)}$ the resulting addition chain causes $O(\sqrt{n})$ operations in $\mathbb{F}_{2^n}$.

- Itoh & Tsujii (1988b), Theorem 2, analyzed the variant when the *binary addition chain* μ is an input. This gives $\lambda_2(n-1) - 1 + \omega_2(n-1) - 1 \leq 2\lambda_2(n-1) - 2 \leq 2\lfloor \log_2(n-1) \rfloor$ multiplications in $\mathbb{F}_{2^n}$. A recursive version of this idea is described in Itoh & Tsujii (1988a).
- In another paper, Asano *et al.* (1989), Theorem 1, used the addition chain generated by the *factor method*[48] which also gives $O(\log(n-1))$ multiplications. ◊

NOTE 6.26. *As a direct consequence of Corollary 6.23 we have the estimate for $q = 2$ on star addition chains*

$$\ell_2(2^m - 1) \leq \ell_2^\star(2^m - 1) \leq \ell_2^\star(m) + m - 1.$$

This bound was first proven by Brauer (1939). Scholz (1937)[49]*, Aufgabe 253, has made a more general conjecture which is known as the (still unproven)* Scholz-Brauer conjecture*:*

$$\ell_2(2^m - 1) \leq \ell_2(m) + m - 1.$$

Parallel computation of the inverse. Our general approach to translate an (original) addition chain for m into a q-addition chain of weight (c_Q, c_A) for $e = \frac{q^m - 1}{q-1}$ works also for parallel addition chains. Let μ be a parallel 2-addition chain of weight $(1, 1)$ for m generated by Algorithm 4.3. Then μ has optimal depth $\lceil \log_2 m \rceil$ and can be run with 2 processors. The derived parallel algorithm for e is the following one.

ALGORITHM 6.27. Computing the inverse in parallel.

Input: A scalar $q \geq 2$ and an exponent $m \geq 1$ with binary representation $(m)_2 = (m_{\lambda-1}, \ldots, m_0)$.

Output: A parallel q-addition chain ε computing $e = \frac{q^m - 1}{q-1}$.

1. Set ε the empty addition chain with $\mathcal{S}(\varepsilon) = \{1\}$.
2. For all processors $0 \leq p < P = 2$ in parallel do 3–9

[48] The *factor method* is described in Knuth (1962), p. 598 and Knuth (1998), Section 4.6.3, p. 463.

[49] Scholz indeed conjectured that $\ell_2(2^m - 1) \leq \ell_2(m-1) + m - 1$ which is not true for $m = 5$ since $\ell_2(4) = 2$ and $\ell_2(31) = 7 > 2 + (5-1)$. But this seems to be just a misprint in the original paper.

3. For $1 \leq i \leq \lceil \log_2 m \rceil$ do 4–9
4. If $p = 0$ and $i < \lambda_2(m)$ then
5. For $1 \leq j \leq 2^{i-1}$ do add the node $(q^{2^{i-1}+j} - q^j)/(q-1)$ and the rule $((q^{2^{i-1}} - 1)/(q-1) \cdot q^{j-1}, -q)$ to ε.
6. Add the node $(q^{2^i} - 1)/(q-1)$ and the rule $(\frac{q^{2^i} - q^{2^{i-1}}}{q-1}, \frac{q^{2^{i-1}} - 1}{q-1})$ to ε.
7. If $p = 1$ and $\sum_{0 \leq j < i} m_j 2^j > 2^{i-1}$ then
8. Set $M_{i-1} = \sum_{0 \leq j < i-1} m_j 2^j$. For $1 \leq j \leq 2^{i-1}$ do add the node $(q^{M_{i-1}+j} - q^j)/(q-1)$ and the rule $((q^{M_{i-1}} - 1)/(q-1) \cdot q^{j-1}, -q)$ to ε.
9. Set $M_i = \sum_{0 \leq j < i-1} m_j 2^j$. Add the node $(q^{M_i} - 1)/(q-1)$ and the rule $(\frac{q^{M_{i-1}+2^{i-1}} - q^{2^{i-1}}}{q-1}, \frac{q^{2^{i-1}} - 1}{q-1})$ to ε.
10. Return ε.

LEMMA 6.28. *Let m and q be positive integers, $q \geq 2$. A parallel q-addition chain for $e = \frac{q^m - 1}{q-1}$ of weight (c_Q, c_A) can be computed using 2 processors in parallel in depth at most*

$$\delta_{q,(c_Q,c_A)}(e) \leq c_A \cdot \lceil \log_2 m \rceil + c_Q \cdot (2^{\lceil \log_2 m \rceil} - 1).$$

Before we proof, this lemma we illustrate the algorithm by an example.

EXAMPLE 6.29. Let $m = 11$ and $e = \frac{q^{11} - 1}{q-1}$ for a scalar $q \in \mathbb{N}_{\geq 2}$. For $(11)_2 = (1011)$ the parallel 2-addition chain generated by Algorithm 4.3 translates as illustrated in Figure 6.3. ◊

PROOF (of Lemma 6.28). Correctness of the algorithm follows by induction on i with the invariant: in lap $1 \leq i < \lambda_2(m)$ of the loop processor 0 computes $\frac{q^{2^i} - 1}{q-1}$, and processor 1 computes in lap $2 \leq i \leq \lceil \log_2(m) \rceil$ the node $\frac{q^{M_{i-1}} - 1}{q-1}$ with $M_i = \sum_{0 \leq j < i} m_j 2^j$ if it is non-zero.

Processor 0 computes 2^{i-1} many q-steps and one non-q-step in lap $1 \leq i < \lambda_2(m)$. Meanwhile, processor 1 computes at most 2^{i-2} many q-steps and one non-q-step if $i \geq 2$. These operations are covered by the operations performed by processor 0. There is a final loop whenever $\lambda_2(m) = \lceil \log_2 m \rceil$. In the final loop, processor 1 calculates at most $2^{\lceil \log_2 m \rceil - 1}$ many q-steps and one non-q-step. This extends the total depth. For both $\lambda_2(m) = \lceil \log_2 m \rceil$ and $\lambda_2(m) = \lceil \log_2 m \rceil + 1$, we get $2^{\lceil \log_2 m \rceil - 1} + \sum_{0 \leq i < \lceil \log_2 m \rceil - 1} 2^i = 2^{\lceil \log_2 m \rceil} - 1$ many q-steps by summation. For $\lambda_2(m) = \lceil \log_2 m \rceil + 1$, we have $\lambda_2(m) - 1 = \lceil \log_2 m \rceil$ non-q-steps. Otherwise, $\lambda_2(m) = \lceil \log_2 m \rceil$ and there are also $\lambda_2(m) - 1 + 1 = \lceil \log_2 m \rceil$ non-q-steps. □

For a normal basis representation we have $c_Q = 0$ and thus the inverse can be computed in parallel with only $P = \max\{2q - 3, 2\}$ processors. For $q = 2$ the

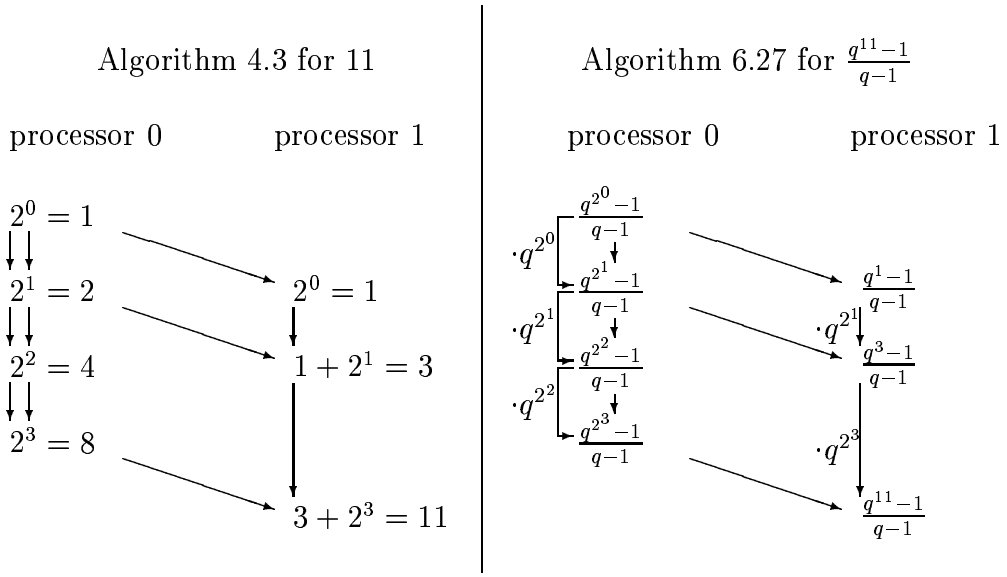


Figure 6.3: Illustration how to the parallel addition chain for 11 translates to a parallel addition chain for $(q^{11}-1)/(q-1)$.

depth is minimal and thus in this special case the number of processors depends no longer on the input size n as it is in Algorithm 4.14.

COROLLARY 6.30. *In a normal basis representation of $\mathbb{F}_{q^n}$, the inverse of an element $B \neq 0$ can be computed*

- *on at most $2q-3$ processors in depth $c_A \cdot (\lceil \log_2(q-1) \rceil + \lceil \log_2(n-1) \rceil + 1)$ if $q > 2$. This bound is at most $2 \cdot c_A$ larger than the optimal depth.*
- *on 2 processors in optimal depth $c_A \cdot \lceil \log_2(n-1) \rceil$ if $q = 2$.*

PROOF. For $q = 2$, the inverse B^{-1} can be computed by evaluating the power $2 \cdot (2^{n-1}-1)$ of B. By Lemma 6.28, we can compute $B^{2^{n-1}-1}$ in $\lambda_2(n-1) = \lceil \log_2(n-1) \rceil$ parallel steps. The final squaring $(B^{2^{n-1}-1})^2$ is free. Since $\omega_2(\frac{2^{n-1}-1}{2-1}) = \omega_2(2^{n-1}-1) = n-1$, we have $\lceil \log_2(n-1) \rceil = \lceil \log_2(\omega_2(\frac{2^{n-1}}{2-1})) \rceil$. By Corollary 4.24 this is optimal!

For $q > 2$, we have to evaluate the power $\frac{q^{n-1}-1}{q-1} \cdot (q-1) \cdot q + (q-2)$. Using $1 \leq P = q' < q$ processors, Algorithm 4.14 computes a digit q' in depth $\lceil \log_2 q' \rceil$. Thus, $q-1$ and $q-2$ can be computed with at most $P = 2q-3$ processors in depth $\lceil \log_2(q-1) \rceil$. The computation of an addition chain for $e = \frac{q^{n-1}-1}{q-1}$ can be done in depth $\lceil \log_2(n-1) \rceil$ on 2 processors according to Lemma 6.28. Thus, we can compute $(B^{q-1})^e$, where $e = \frac{q^{n-1}-1}{q-1}$, in depth $\lceil \log_2(q-1) \rceil + \lceil \log_2(n-1) \rceil$. Evaluating the q-th power of $B^{q^{n-1}-1}$ is free. We add $q-2$ in a final non-q-step.

This gives a total depth of $\lceil \log_2(q-1) \rceil + \lceil \log_2(n-1) \rceil + 1$. The optimal depth is $\delta_{2,(0,1)}(q^n-2) = \lceil \log_2 \omega_q(q^n-2) \rceil$ with $\omega_q(q^n-2) = \omega_q((\underbrace{q-1, \ldots, q-1}_{n-1}, q-2)) =$ $(n-1)(q-1) + (q-2) = n(q-1) - 1$. Therefore, the depth is

$$\begin{aligned} & \lceil \log_2(n(q-1)-1) \rceil \geq \log_2(n \cdot (q-1) - 1) \geq \log_2(n(q-1)) - 1 \\ = \; & \log_2 n + \log_2(q-1) - 1 \geq \lceil \log_2 n \rceil + \lceil \log_2(q-1) \rceil - 3. \end{aligned}$$

The latter one is the depth of our parallel inversion algorithm plus 2. □

Experiments. In theory, the approach of Itoh & Tsujii (1988b) and the parallel inversion described by Algorithm 6.27 differ only by a constant of at most 2. Is it worth while to insert a shorter addition chain for $n-1$ than the binary addition chain if possible? We give an answer that is based on experiments. Again, we used the optimal normal bases of the test series *Normal*. Multiplication and squaring have been described in Section 6.3 above. For each field extension $\mathbb{F}_{2^n}$ of the test series *Normal*, we ran four different versions as documented in Table A.13. We compared the parallel inversion (Algorithm 6.27) with the sequential version which is basically Algorithm 6.21. For the sequential algorithm, we chose three different addition chains for $n-1$. For each value of n, we computed the inverse of 100 randomly chosen elements in $\mathbb{F}_{2^n}^{\times}$. The times as given in Figure 6.4 are the average of these 100 trials. The binary addition chain as input for the sequential algorithm was suggested by Itoh & Tsujii (1988b). The corresponding times are marked by red dots and labeled *binary a.c.* in Figure 6.4. A second star addition chain for $n-1$ is due to Brauer's idea, but with the modifications described in Section 3.2. This generalization of the binary addition chain is labeled *Brauer a.c.* in Figure 6.4. It shows slightly better times for 18 values of n out of the chosen 50 in the test series *Normal*. It is always at least as good as the binary addition chain (up to accuracy of measurement). The quotient between the times is almost the same as the quotient between the length of the two addition chains. Thus, a shorter chain means faster inversion (compare the quotients of columns 3 to 5 and 4 to 6, respectively, of Table A.13). In case of a Brauer star addition chain, this causes a maximal speed-up of 1.13 for $n = 1018$. On average, we get a speed-up of 1.03 over all 50 values for n.

Knuth (1962, 1998) described a method called *power tree* to generate short addition chains. Our third sequential version used such a chain. The times in Figure 6.4 are labeled *power tree a.c.* and are marked by blue dots. We count 41 shorter addition chains for the values of the test series *Normal* compared to the binary addition chain. The maximal speed-up is here 1.27 for $n = 1798$; the time for inversion in $\mathbb{F}_{2^n}$ is only about 80% compared to the original algorithm of Itoh & Tsujii (1988b) for three other values $n \in \{5598, 6396, 7803\}$. Since the power tree generates significantly smaller addition chains than the binary method, we have an average speed-up of 1.12 for the test series *Normal*. We conclude that

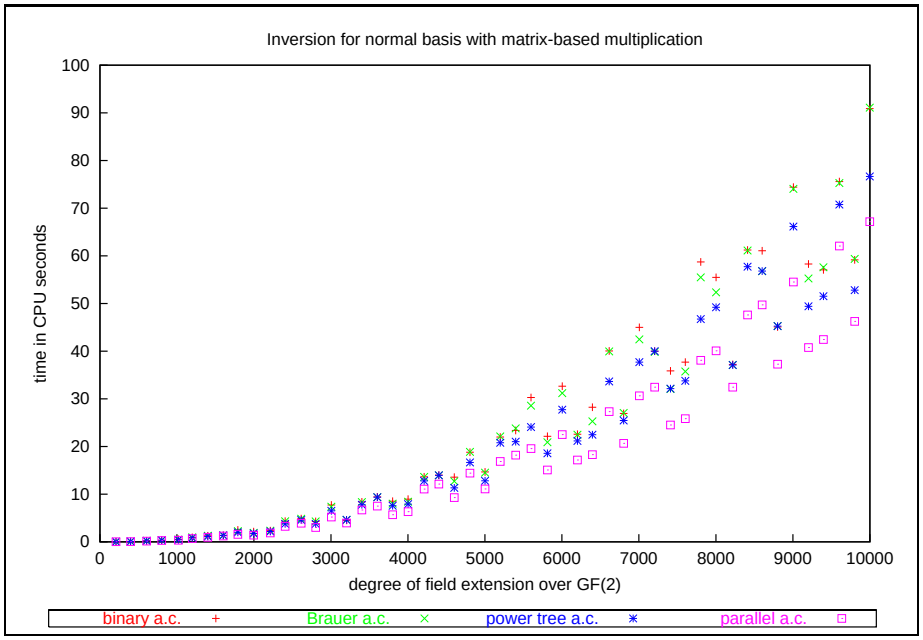


Figure 6.4: Times for inversion for the test series *Normal*. The times are the average of 100 trials. The graphical representation compares the sequential inversion algorithm a la Fermat (Algorithm 6.21) with the parallel version (□) given in Algorithm 6.27. The sequential version was run on three different addition chains μ for $m = n - 1$ as input: the binary addition chain (+), Brauer's addition chain (×), and the addition chain generated by the power tree algorithm (∗).

substituting the binary addition chain by a shorter (or even a shortest) one for $n - 1$ in Algorithm 6.21 is a successful way to speed up inversion if it is based on Fermat's Little Theorem 2.3 in $\mathbb{F}_{2^n}$.

For Algorithm 6.27, a parallel addition chain of optimal depth for $n - 1$ is hardwired in the algorithm. Figure 6.4 shows that this parallel approach—labeled *parallel a.c.*—gives always the shortest measured times for inversion in $\mathbb{F}_{2^n}$. It beats the fastest sequential addition chain for all values. We reach an average speed-up of 1.35 over all 50 values. We get a speed-up of at least 1.5 compared to the binary addition chain for 7 values of n; these are $n \in \{1018, 1791, 1996, 3802, 5598, 6396, 7803\}$.

Conclusions. We have generalized an approach of Itoh & Tsujii (1988b) to apply sequential as well as parallel exponentiation to inversion in a normal basis representation of $\mathbb{F}_{q^n}$. Both the theoretical fact and our experimental observations show that normal bases support the parallel computing in the case of exponentiation. We have illustrated this for parallel inversion in $\mathbb{F}_{q^n}$. Nevertheless, the slow matrix-based multiplication is the Achilles' Heel of this data structure! Thus, our goal will be to speed up the multiplication in a normal basis representation. To

do so, we restrict to a special type of normal elements in the subsequent sections.

7. Gauß periods

In the preceding Section 6, we have considered arbitrary normal elements. We have shown that this basis representation supports parallel exponentiation. But we have also detected that matrix-based multiplication is the bottleneck that prevents good performance.

Thus, we have to speed up multiplication for normal bases. Our idea is to replace matrix-based multiplication by faster algorithms for specific normal elements. Our choice for these specific normal elements are Gauß periods which were used for fast arithmetic by the construction of optimal normal bases. The normal bases given in Fact 6.13 are just *prime Gauß periods of type* (n, k) *over* $\mathbb{F}_q$ with $k \in \{1, 2\}$. A reviewer remarked the relation first, as cited in Ash *et al.* (1989)[50].

In his *Disquisitiones Arithmeticae* in Article 343, Gauß (1801) introduced certain elements—which are nowadays called Gauß periods—for his famous solution of the problem how to construct a regular 17-gon by ruler and compass. Due to this application, Gauß was only interested in the prime case. But he already remarked at the end of Article 356 that this method can be generalized.[51]

Normal elements which are Gauß periods, also denoted as normal Gauß periods, are connected to *cyclotomic polynomials*. This observation will point a way to replace matrix-based multiplication by polynomial multiplication. The same idea was already described for some special cases by Gao *et al.* (1995) and Gao *et al.* (2000) for prime Gauß periods over $\mathbb{F}_q$, and by Blake *et al.* (1998) for optimal normal bases in $\mathbb{F}_{2^n}$. Both cases are covered by our new and more general results.

In this section, we present Gauß periods and some of their main properties for further use. In the first part (Section 7.1), we give the definition of generalized Gauß periods by Feisel *et al.* (1999). The second part (Section 7.2) is devoted to certain properties of these generalized Gauß periods. In particular, we are interested in the case when Gauß periods generate normal bases. Furthermore, we discuss the relation between normal Gauß periods in field towers and the canonical projection of the subgroup that defines a Gauß period. In Section 7.3, we give the connection of Gauß periods to cyclotomic polynomials. In particular, we want to use this relation as our key to replace matrix-based multiplication by polynomial multiplication. In the final part (Section 7.4), we collect some properties on arbitrary normal elements, i.e. of elements α generating a normal basis $\mathcal{N} = (\alpha, \ldots, \alpha^{q^{n-1}})$ in $\mathbb{F}_{q^n}$.

[50] "One of the reviewers of this paper pointed out that the element α in this theorem is of classical origin and is referred as a period of Gauss.", Ash *et al.* (1989), p. 196.

[51] "These theorems retain the same or even greater elegance when they are extended to composite values of n. But these matters are on a higher level of investigation, and we will reserve their consideration for another occasion.", Gauß (1801), Article 356. [Gauß' n is the r used above.]

7.1. Definition of general Gauß periods. We fix some notation.

NOTATION 7.1. *Let q be a prime power, r be an integer greater than 1, and let n and k be positive integers, such that q and r are co-prime and $\phi(r) = nk$. Here $\phi(r) = \#\{a \in \mathbb{N}_{\geq 1} : 1 \leq a < r, \gcd(a, r) = 1\}$ denotes* Euler's totient function. *Let ζ be a primitive r-th root of unity in an extension field of $\mathbb{F}_q$.*

Furthermore, let $r_1 \cdots r_t$ with $r_i = p_i^{e_i}$ for $1 \leq i \leq t$ be the *prime power decomposition* of r. We set $R_1 = \prod_{1 \leq i \leq t, e_i = 1} p_i$ the *square-free part*[52] of r and $R_2 = r/R_1$ the *non-squarefree part.* A Gauß period α in $\mathbb{F}_{q^n}$ is given as a sum of powers of a primitive r-th root of unity ζ. Feisel *et al.* (1999) introduced the following generalization of Gauß' definition.

DEFINITION 7.2. *Let r, q, n, k be positive integers, and let R_1, R_2 be the square-free and non-squarefree part of r, respectively. We set*

$$b(x) = x^{R_2} \cdot \prod_{\substack{1 \leq i \leq t \\ p_i \mid R_2}} \sum_{1 \leq s \leq e_i} x^{r/p_i^s} \in \mathbb{F}_q[x]. \tag{7.3}$$

Let $\mathcal{K}$ be a subgroup of $\mathbb{Z}_r^\times$ of order k, and ζ be a primitive r-th root of unity in an extension field of $\mathbb{F}_q$. A (general) Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ *with respect to ζ is defined as*

$$\alpha = \sum_{a \in \mathcal{K}} b(\zeta^a).$$

The definition contains the power ζ^a where a is in $\mathbb{Z}_r^\times$. But this is well-defined, since ζ is a primitive r-th root of unity. Let a and a' be two integers such that $a \equiv a' \bmod r$. Then we have $a - a' = ur$ for an $u \in \mathbb{Z}$ and $\zeta^{a-a'} = \zeta^{ur} = 1^u = 1$, i.e. $\zeta^a = \zeta^{a'}$. Although, ζ is a root of the polynomial $x^r - 1 \in \mathbb{F}_q[x]$ and hence an element in $\mathbb{F}_{q^r}$, the element α is in the smaller field $\mathbb{F}_{q^n}$. To see why, we remark that there are at most n distinct cosets in $q^0\mathcal{K}, \ldots, q^{n-1}\mathcal{K}$ of $\mathcal{K}$, see also Fact 7.6. But then $q^n\mathcal{K} = \mathcal{K}$, and

$$\alpha^{q^n} = \sum_{a \in \mathcal{K}} b(\zeta^a)^{q^n} = \sum_{a \in \mathcal{K}} b(\zeta^{aq^n}) = \sum_{a \in q^n\mathcal{K}} b(\zeta^a) = \sum_{a \in \mathcal{K}} b(\zeta^a) = \alpha.$$

For $r = p$, we have $r = R_1$ and $R_2 = 1$, and the polynomial $b(x)$ in (7.3) collapses to $b(x) = x^1$. The group $\mathbb{Z}_p^\times$ is cyclic, and a subgroup $\mathcal{K}$ is therefore uniquely determined by its order k. Hence, the already mentioned *prime Gauß period of type (n, k) over $\mathbb{F}_q$* is

$$\alpha = \sum_{a \in \mathcal{K}} \zeta^a.$$

[52] Our definition follows the one in Feisel *et al.* (1999). It differs from the common definition of the square-free part. Usually, the square-free part of $r = \prod_{1 \leq i \leq t} p_i^{e_i}$ is defined as $\prod_{1 \leq i \leq t} p_i$, see e.g. von zur Gathen & Gerhard (1999), Section 14.6.

For our work, the case where $r = p^e$ is a prime power with $e > 1$ is important. Here $r = R_2$ and $R_1 = 1$ and (7.3) transforms into $b(x) = x^r \cdot \sum_{1 \le s \le e} x^{p^{e-s}}$. Since ζ is a primitive r-th root of unity, i.e. $\zeta^r = 1$, the *prime power Gauß period of type* $(n, \mathcal{K})$ *over* $\mathbb{F}_q$ is

$$\alpha = \sum_{a \in \mathcal{K}} \sum_{0 \le s < e} \zeta^{p^s a}.$$

EXAMPLE 7.4. Let q be equal to two.

(i) Let $r = 5$, ζ a primitive 5-th root of unity, and let $\mathcal{K} = \{1\}$ be the uniquely determined subgroup of $\mathbb{Z}_5^\times$ of order $k = 1$. Then $\alpha = \zeta$ is a *prime Gauß period* of type $(4, 1)$ over $\mathbb{F}_2$.

(ii) Let $r = 3^2$, ζ a primitive 9-th root of unity, and $\mathcal{K} = \{1, 8\}$ be the uniquely determined subgroup of $\mathbb{Z}_9^\times$ of order $k = 2$. Then $\alpha = \zeta^{1\cdot 1} + \zeta^{3\cdot 1} + \zeta^{1\cdot 8} + \zeta^{3\cdot 8} = \zeta + \zeta^3 + \zeta^8 + \zeta^6$ is a *prime power Gauß period* of type $(3, \{1, 8\})$ over $\mathbb{F}_2$. Here we used $\zeta^{24} = \zeta^6$ (since $24 \equiv 6 \bmod 9$).

(iii) Let $r = 3^2 \cdot 5$ and ζ be a primitive 45-th root of unity. There are three different subgroups of order $k = 2$ of $\mathbb{Z}_{45}^\times$ which define three different *general Gauß periods*. The subgroup $\mathcal{K}_1 = \{1, 26\}$ determines the Gauß period $\alpha_1 = \zeta^{14} + \zeta^{24} + \zeta^4 + \zeta^{39}$ of type $(12, \{1, 26\})$. The subgroup $\mathcal{K}_2 = \{1, 44\}$ generates the Gauß period $\alpha_2 = \zeta^{14} + \zeta^{24} + \zeta^{21} + \zeta^{31}$. Finally, $n = 12$ and $\mathcal{K}_3 = \{1, 19\}$ define the Gauß period $\alpha_3 = \zeta^{14} + \zeta^{24} + \zeta^6 + \zeta^{41}$. ◊

A Gauß period is not completely determined by its parameters q, n and $\mathcal{K}$. It also depends on the chosen primitive r-th root of unity ζ. Let $\alpha = \sum_{a \in \mathcal{K}} b(\zeta^a)$ be a Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ with respect to ζ where $\mathcal{K}$ is a subgroup of $\mathbb{Z}_r^\times$. Let $\zeta' \in \mathbb{F}_{q^r}$ be another primitive r-th root of unity and $\alpha' = \sum_{a \in \mathcal{K}} b(\zeta'^a)$ be the related Gauß period. By construction there is an $a' \in \mathbb{Z}_r^\times$ such that $\zeta' = \zeta^{a'}$. If we assume $\{cq^h \colon c \in \mathcal{K}, h \in \mathbb{Z}\} = \mathbb{Z}_r^\times$ then there are $c \in \mathcal{K}$ and $0 \le h < n$ such that $a' = cq^h$ and $\zeta' = \zeta^{a'} = \zeta^{cq^h}$. This gives

$$\alpha' = \sum_{a \in \mathcal{K}} b(\zeta'^a) = \sum_{a \in \mathcal{K}} b(\zeta^{acq^h}) = \left(\sum_{a \in \mathcal{K}} b(\zeta^a) \right)^{q^h} = \alpha^{q^h}.$$

Therefore, two distinct primitive r-th roots of unity define two conjugate normal elements. We do not emphasize the specification of the primitive root of unity ζ if we are not interested in the concrete conjugate of a Gauß period.

7.2. The condition $\langle q, \mathcal{K} \rangle = \mathbb{Z}_r^\times$. Subsequently, we restrict to Gauß periods of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ with $\langle q, \mathcal{K} \rangle = \mathbb{Z}_r^\times$. Here $\langle q, \mathcal{K} \rangle = \{q^h a \bmod r \colon h \in \mathbb{Z}, a \in \mathcal{K}\}$ denotes the subgroup of $\mathbb{Z}_r^\times$ that is jointly generated by powers of $(q \bmod r)$ and the subgroup $\mathcal{K}$. The Main Theorem of Feisel *et al.* (1999) states that this condition is equivalent to the fact that a Gauß period is normal.

NORMAL GAUSS PERIOD THEOREM 7.5 (Feisel *et al.* 1999). *Let α be a Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ with $\mathcal{K}$ a subgroup of $\mathbb{Z}_r^\times$. Then α is normal in $\mathbb{F}_{q^n}$ if and only if $\langle q, \mathcal{K}\rangle = \mathbb{Z}_r^\times$.*

EXAMPLE 7.4 CONTINUED. (i) Since $\langle 2, \{1\}\rangle = \{2, 4, 3, 1\} = \mathbb{Z}_5^\times$, the Gauß period of type $(4, \{1\})$ is normal in $\mathbb{F}_{16}$ over $\mathbb{F}_2$.

(ii) One can easily check that $\langle 2, \{1, 8\}\rangle = \mathbb{Z}_9^\times$. Hence, the Gauß period of type $(3, \{1, 8\})$ is normal in $\mathbb{F}_8$ over $\mathbb{F}_2$.

(iii) Only the two subgroups $\mathcal{K}_1 = \{1, 26\}$ and $\mathcal{K}_2 = \{1, 44\}$ generate normal Gauß periods in $\mathbb{F}_{2^{12}}$ over $\mathbb{F}_2$. For $\mathcal{K}_3 = \{1, 19\}$ we have $\langle 2, \{1, 19\}\rangle = \{2, 38, 4, 31, 8, 17, 16, 34, 23, 32, 1, 19\} \neq \mathbb{Z}_{45}^\times$. Thus, the Gauß period of type $(12, \{1, 19\})$ over $\mathbb{F}_2$ is not normal in $\mathbb{F}_{4096}$. ◊

Throughout this section we will assume that all parameters n, q, r and a subgroup $\mathcal{K}$ of $\mathbb{Z}_r^\times$ of order $k = \frac{\phi(r)}{n}$ are given such that $\phi(r) = nk$ and $\langle q, \mathcal{K}\rangle = \mathbb{Z}_r^\times$. Usually, only n and q are given. We will discuss an algorithmic approach how to find r and $\mathcal{K}$ in this case in Section 9.4.

A necessary condition. For our arithmetic purposes we derive some further conclusions from the condition $\langle q, \mathcal{K}\rangle = \mathbb{Z}_r^\times$. The sets $q^h\mathcal{K}$, with $0 \leq h < n$, are (right) cosets of the subgroup $\mathcal{K}$. This implies the following facts.

FACT 7.6. *Let $\mathcal{K}$ be a subgroup of $\mathbb{Z}_r^\times$ order k and $n = \phi(r)/k$. If $\langle q, \mathcal{K}\rangle = \mathbb{Z}_r^\times$ then*

(i) for all $i, j \in \mathbb{Z}$ we have $q^i\mathcal{K} \cap q^j\mathcal{K} \neq \emptyset$ if and only if $q^i\mathcal{K} = q^j\mathcal{K}$,

(ii) $\biguplus_{0\leq h<n} q^h\mathcal{K} = \langle q, \mathcal{K}\rangle$ is a partition of $\mathbb{Z}_r^\times$, and

(iii) $q^n \in \mathcal{K}$.

As a direct consequence, we have that $\langle q, \mathcal{K}\rangle = \mathbb{Z}_r^\times$ is a necessary condition for a Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ to be normal.

REMARK 7.7. *Let $r \in \mathbb{N}_{\geq 2}$, and let α be a general Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ with $\mathcal{K} \subseteq \mathbb{Z}_r^\times$. If α is normal in $\mathbb{F}_{q^n}$ then $\langle q, \mathcal{K}\rangle = \mathbb{Z}_r^\times$.*

PROOF. Assuming the contrary, we may suppose that $\langle q, \mathcal{K}\rangle \neq \mathbb{Z}_r^\times$. Then there is an $0 \leq i < j < n$ such that $q^i\mathcal{K} = q^j\mathcal{K}$, and we get

$$\alpha^{q^i} = \left(\sum_{a\in\mathcal{K}} b(\zeta^a)\right)^{q^i} = \sum_{a\in\mathcal{K}} b(\zeta^{aq^i}) = \sum_{a\in q^i\mathcal{K}} b(\zeta^a)$$

$$= \sum_{a\in q^j\mathcal{K}} b(\zeta^a) = \sum_{a\in\mathcal{K}} b(\zeta^{aq^j}) = \left(\sum_{a\in\mathcal{K}} b(\zeta^a)\right)^{q^j} = \alpha^{q^j}.$$

Therefore, $\alpha, \ldots, \alpha^{q^{n-1}}$ are not linearly independent which is a contradiction. □

The canonical projection. Let r' be a positive divisor of r not equal to 1, and let

$$\pi_{r'} \colon \mathbb{Z}_r^\times \to \mathbb{Z}_{r'}^\times \text{ with } \pi_{r'}(a) = (a \bmod r') \tag{7.8}$$

the *canonical projection* of $\mathbb{Z}_r^\times$ onto $\mathbb{Z}_{r'}^\times$ which is a surjective epimorphism on groups. We apply this canonical projection to the subgroup $\mathcal{K} \subseteq \mathbb{Z}_r^\times$ that defines a Gauß period of type $(n, \mathcal{K})$ with respect to ζ, and we get the image $\pi_{r'}(\mathcal{K})$ of the subgroup $\mathcal{K}$ under this epimorphism. Thus, $\pi_{r'}(\mathcal{K})$ is a subgroup of $\mathbb{Z}_{r'}^\times$. The order k' of $\pi_{r'}(\mathcal{K})$ divides both $k = \#\mathcal{K}$ and $\phi(r') = \#\mathbb{Z}_{r'}^\times$, and $n' = \phi(r')/k'$ is a positive integer. The following lemma states that the canonical projection gives a normal Gauß period in a subfield of $\mathbb{F}_{q^n}$.

Lemma 7.9. *Let α be a Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ with respect to ζ. Let r' be a positive divisor of r greater than 1, and let $\pi_{r'}$ be as in (7.8). Set $k' = \#\pi_{r'}(\mathcal{K})$ and $n' = \phi(r')/k'$ as above. If $\langle q, \mathcal{K}\rangle = \mathbb{Z}_r^\times$ then n' divides n, and the Gauß period α' of type $(n', \pi_{r'}(\mathcal{K}))$ over $\mathbb{F}_q$ with respect to $\zeta^{r/r'}$ satisfies $\langle q, \pi_{r'}(\mathcal{K})\rangle = \mathbb{Z}_{r'}^\times$.*

The proof—its main ideas are sketched in Figure 7.1—uses the following observation about the Galois group of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$. Wassermann (1993), Bemerkung 3.1.2, proved this remark for prime Gauß periods.

Remark 7.10. *Let α be a normal Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$. Then the quotient group $\mathbb{Z}_r^\times/\mathcal{K}$ is isomorphic to the Galois group* $\mathrm{Gal}(\mathbb{F}_{q^n}/\mathbb{F}_q)$.

Proof. Since α is normal, we have $\langle q, \mathcal{K}\rangle = \mathbb{Z}_r^\times$ by Remark 7.7. Let $\bar{}\colon \mathbb{Z}_r^\times \to \mathbb{Z}_r^\times/\mathcal{K}$ be the canonical projection onto the quotient group. Since $\langle q, \mathcal{K}\rangle = \mathbb{Z}_r^\times$, we have $\biguplus_{0 \le h < n} q^h\mathcal{K}$ a partition of $\mathbb{Z}_r^\times$ by Fact 7.6(ii) and $\mathbb{Z}_r^\times/\mathcal{K} = \{\bar{1}, \bar{q}, \ldots, \bar{q}^{n-1}\} = \langle \bar{q}\rangle$. Hence, the quotient group is cyclic of order n.

The Galois group $\mathrm{Gal}(\mathbb{F}_{q^n}/\mathbb{F}_q)$ has order $[\mathbb{F}_{q^n} : \mathbb{F}_q] = n$, and it is cyclic with the Frobenius automorphism $\sigma(A) = A^q$ as its canonical generator as stated in Fact 2.9. But two cyclic groups of same order are isomorphic. The isomorphism is given by mapping $\bar{q}^h \in \mathbb{Z}_r^\times/\mathcal{K}$ on $\sigma^h \in \mathrm{Gal}(\mathbb{F}_{q^n}/\mathbb{F}_q)$ for $0 \le h < n$. □

Proof (of Lemma 7.9). The canonical projection $\pi_{r'}$ is an epimorphism, and therefore $\langle \pi_{r'}(q), \pi_{r'}(\mathcal{K})\rangle = \mathbb{Z}_{r'}^\times$ if $\langle q, \mathcal{K}\rangle = \mathbb{Z}_r^\times$. It remains to show that $\mathbb{F}_{q^{n'}}$ can be regarded as a subfield of $\mathbb{F}_{q^n}$ to complete the proof.

Set $\psi_{r'} = {}^{-} \circ \pi_{r'}$ which is an epimorphism from $\mathbb{Z}_r^\times$ onto the quotient group $\mathbb{Z}_{r'}^\times/\pi_{r'}(\mathcal{K})$. For all $a \in \mathcal{K}$, we have $\psi_{r'}(a) = \overline{\pi_{r'}(a)} \in \overline{\pi_{r'}(\mathcal{K})} = \bar{1}$, and hence $\mathcal{K} \subseteq \ker \psi_{r'}$. We can decompose the canonical projection from $\mathbb{Z}_r^\times$ onto $\mathbb{Z}_r^\times/\ker\psi_{r'}$ as $\psi' \circ {}^{-}$, where ψ' maps $\mathbb{Z}_r^\times/\mathcal{K}$ onto $\mathbb{Z}_r^\times/\ker\psi_{r'}$. The *fundamental theorem of*

homomorphisms on groups (see e.g. Jacobson (1974), p. 60) claims the existence of an isomorphism $\bar{\psi}\colon \mathbb{Z}_r^\times/\ker\psi_{r'} \to \mathbb{Z}_{r'}^\times/\pi_{r'}(\mathcal{K})$. By combining this with the canonical projection $\psi'\colon \mathbb{Z}_r^\times/\mathcal{K} \to \mathbb{Z}_{r'}^\times/\ker\psi_{r'}$, we get the epimorphism $\bar{\pi}_{r'} = \bar{\psi}\circ\psi'$ from $\mathbb{Z}_r^\times/\mathcal{K}$ onto $\mathbb{Z}_{r'}^\times/\pi_{r'}(\mathcal{K})$. Since $\bar{\pi}_{r'}$ is surjective, we have $n' = \#\mathbb{Z}_{r'}^\times/\pi_{r'}(\mathcal{K})$ a divisor of $n = \#\mathbb{Z}_r^\times/\mathcal{K}$. By Remark 7.10, the Galois group $\mathrm{Gal}(\mathbb{F}_{q^{n'}}/\mathbb{F}_q)$ is a subgroup of $\mathrm{Gal}(\mathbb{F}_{q^n}/\mathbb{F}_q)$. Thus, $\mathbb{F}_{q^{n'}}$ is a subfield of $\mathbb{F}_{q^n}$. This completes the proof of Lemma 7.9. □

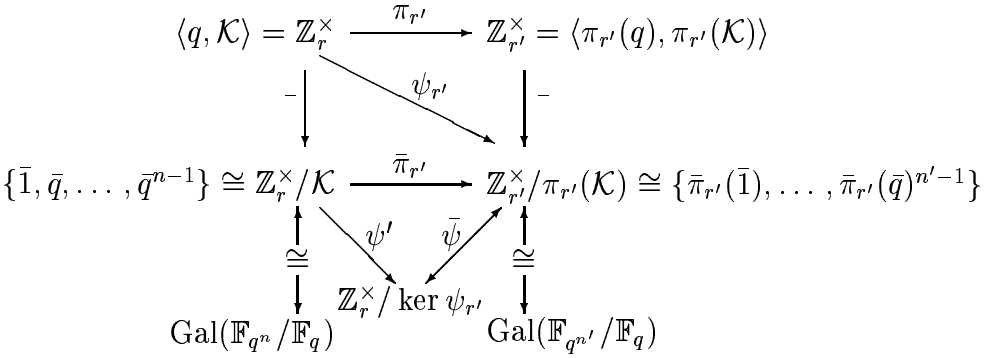


Figure 7.1: Illustration of the different projections discussed in this paragraph.

The connection between the group $\mathbb{Z}_r^\times$ and the normal Gauß period in a subfield plays an important rôle in what follows. We illustrate this in the case of prime power Gauß periods. Let r be a prime power p^e with $e \geq 2$, and let ζ be a primitive p^e-th root of unity. We suppose that the subgroup $\mathcal{K}$ of $\mathbb{Z}_r^\times$ defines a normal Gauß period $\alpha = \sum_{a\in\mathcal{K}}\sum_{0\leq s<e}\zeta^{ap^s}$ of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ with respect to ζ. Then $\langle q, \mathcal{K}\rangle = \mathbb{Z}_{p^e}^\times$ by Remark 7.7. For $0 < \ell < e$, we have $\zeta_\ell = \zeta^{p^{e-\ell}}$ a primitive p^ℓ-th root of unity, and we set $n_\ell = \phi(p^\ell)/\#\pi_{p^\ell}(\mathcal{K})$. Then

$$\alpha_\ell = \sum_{a\in\pi_{p^\ell}(\mathcal{K})}\sum_{0\leq s<\ell}\zeta_\ell^{ap^s}$$

is the Gauß period of type $(n_\ell, \pi_{p^\ell}(\mathcal{K}))$ over $\mathbb{F}_q$ with respect to ζ_ℓ by Lemma 7.9. Since $\langle q, \pi_{p^\ell}(\mathcal{K})\rangle = \mathbb{Z}_{p^\ell}^\times$, the Gauß period α_ℓ is normal in $\mathbb{F}_{q^{n_\ell}}$ over $\mathbb{F}_q$.

Example 7.11. Let $q = 2$ and $r = 9$. Then the subgroup $\mathcal{K} = \{1, 8\} \subseteq \mathbb{Z}_9^\times$ and a primitive 9-th root of unity ζ jointly define the normal Gauß period $\alpha = \zeta+\zeta^3+\zeta^8+\zeta^6$ of type $(3, \{1, 8\})$ over $\mathbb{F}_2$. The canonical projection $\pi_3\colon \mathbb{Z}_9^\times \to \mathbb{Z}_3^\times$ maps $\mathcal{K}$ onto the subgroup $\pi_3(\mathcal{K}) = \{1, 2\}$ of $\mathbb{Z}_3^\times$. We have that $\zeta_1 = \zeta^{3^{2-1}} = \zeta^3$ is a primitive 3-rd root of unity. Lemma 7.9 claims that $\alpha_1 = \sum_{a\in\pi_3(\mathcal{K})}\zeta_1^a = \zeta^3+\zeta^6$ is a normal Gauß period of type $(1, \{1, 2\})$ over $\mathbb{F}_2$. In fact, we have $\langle 2, \{1, 2\}\rangle = \mathbb{Z}_3^\times$, and α_1 is indeed a normal prime Gauß period. ◊

7.3. Cyclotomic polynomials. Primitive roots of unity are related to a special class of polynomials: the *cyclotomic polynomials*. The proofs of the following facts are given in Lidl & Niederreiter (1983), Section 2.4.

DEFINITION 7.12. *Let q be a prime power, and r be a positive integer co-prime to q. Let ζ be a primitive r-th root of unity over $\mathbb{F}_q$. Then the polynomial*

$$\Phi_r = \prod_{\substack{0<s<r \\ \gcd(s,r)=1}} (x - \zeta^s)$$

is called the r-th cyclotomic polynomial *over* $\mathbb{F}_q$.

Since the roots of Φ_r are all $\phi(r)$ distinct primitive r-th roots of unity, the degree of Φ_r is just $\deg \Phi_r = \phi(r)$. Therefore, we have $\zeta \in \mathbb{F}_{q^{\phi(r)}}$.

FACT 7.13 (Lidl & Niederreiter 1983, Theorem 2.45). *Let q and r be as above. Then*

(i) $x^r - 1 = \prod_{1 \le d \le r,\, d|r} \Phi_d$, *and*

(ii) the coefficients of Φ_r *belong to the prime subfield of* $\mathbb{F}_q$.

If $r = p^e$ is a prime power then the formula above implies

$$\Phi_p(x) = \frac{x^p - 1}{x - 1} = \sum_{0 \le i < p} x^i, \text{ and} \tag{7.14}$$

$$\Phi_{p^e}(x) = \Phi_{p^\ell}(x^{p^{e-\ell}}) = \frac{x^{p^e} - 1}{x^{p^{e-\ell}} - 1} = \sum_{0 \le i < p^\ell} x^{ip^{e-\ell}} \text{ for } 0 < \ell \le e, \text{ and} \tag{7.15}$$

$$x^{p^e} - 1 = (x - 1) \cdot \prod_{1 \le \ell \le e} \Phi_{p^\ell}(x). \tag{7.16}$$

Over the field $\mathbb{Q}$ of rational numbers, the cyclotomic polynomial Φ_r is always irreducible. This is no longer true in the case of a finite field $\mathbb{F}_q$ with non-zero characteristic. But in this case the factorization pattern is well-known.

FACT 7.17 (Lidl & Niederreiter 1983, Theorem 2.47). *Let q be a prime power co-prime to a positive integer r, and let $N = \operatorname{ord}_r(q)$ be the order of q in $\mathbb{Z}_r^\times$. Then the r-th cyclotomic polynomial $\Phi_r \in \mathbb{F}_q[x]$ factors into $\frac{\phi(r)}{N}$ distinct monic irreducible polynomials of the same degree N.*

We denote the $d = \frac{\phi(r)}{N}$ irreducible factors by $\mu_1, \ldots, \mu_d \in \mathbb{F}_q[x]$. By the *Chinese Remainder Theorem* we have the ring isomorphism

$$\chi' : \begin{array}{rcl} \mathcal{R} = \mathbb{F}_q[x]/(\Phi_r) & \to & \mathbb{F}_q[x]/(\mu_1) \times \cdots \times \mathbb{F}_q[x]/(\mu_d) \\ A & \mapsto & (A \bmod \mu_1, \ldots, A \bmod \mu_d). \end{array} \tag{7.18}$$

Since $\Phi_r(\zeta) = 0$ for any primitive r-th root of unity $\zeta \in \mathbb{F}_{q^{\phi(r)}}$, we know that the minimal polynomial μ_ζ of ζ in $\mathbb{F}_q[x]$ is one of the $\mu_1, \ldots, \mu_d$. Then

$$\varphi_\zeta \colon \mathbb{F}_q(\zeta) \;\to\; \mathbb{F}_q[x]/(\mu_\zeta) \text{ with } \varphi_\zeta\left(\sum_{0\le i<N} A_i\zeta^i\right) = \sum_{0\le i<N} A_i(x^i \bmod \mu_\zeta)$$

is the canonical isomorphism between the two images of $\mathbb{F}_{q^N}$. The field $\mathbb{F}_q(\alpha)$—which is constructed by adjoining the Gauß period $\alpha = \sum_{a\in\mathcal{K}} b(\zeta^a)$ to $\mathbb{F}_q$—is a subfield of $\mathbb{F}_q(\zeta)$. Thus, we know the image of α in $\mathbb{F}_q[x]/(\mu_\zeta)$. The key for fast multiplication of Gauß periods lies in the choice of a suitable preimage of α in $\mathcal{R}$.

REMARK 7.19. *Let q, r, n, k be as above, and let $\mathcal{K}$ be a subgroup of $\mathbb{Z}_r^\times$ of order $k = \phi(r)/n$ such that $\langle q, \mathcal{K}\rangle = \mathbb{Z}_r^\times$. Let $\mu \in \mathbb{F}_q[x]$ be an irreducible factor of Φ_r and $\zeta \in \mathbb{F}_{q^{nk}}$ a primitive r-th root of unity. Then there is an element $a \in \mathcal{K}$ such that $\mu(\zeta^a) = 0$.*

PROOF. Since $\langle q, \mathcal{K}\rangle = \mathbb{Z}_r^\times$ and $\biguplus_{0\le h<n} q^h\mathcal{K}$ is a partition of $\mathbb{Z}_r^\times$, we have $\{\zeta^{aq^h} \colon a \in \mathcal{K} \text{ and } 0 \le h < n\}$ the set of all distinct primitive r-th roots of unity. Thus, there are $a \in \mathcal{K}$ and $h \in \{0, \ldots, n-1\}$ such that $\mu(\zeta^{aq^h}) = 0$. But then also $0 = \mu(\zeta^{aq^h})^{q^{n-h}} = \mu(\zeta^{aq^n}) = \mu(\zeta^{a'})$ where $a' = q^n a$. We have $\zeta^{a'}$ a primitive r-th root of unity of the claimed form since $q^n \in \mathcal{K}$ by Fact 7.6(iii), and $a' \in \mathcal{K}$ by $a \in \mathcal{K}$. □

Let $\mu \in \mathbb{F}_q[x]$ be an irreducible factor of Φ_r, and let $c \in \mathcal{K}$ such that $\zeta' = \zeta^c$ is a root of μ. Then we have for $c \in \mathcal{K}$

$$\alpha = \sum_{a\in\mathcal{K}} b(\zeta^a) = \sum_{a\in\mathcal{K}} b(\zeta^{cc^{-1}a}) = \sum_{a\in\mathcal{K}} b(\zeta'^a) = \alpha,$$

and $\varphi_{\zeta'}(\alpha) = \sum_{a\in\mathcal{K}} b(x^a \bmod \mu_{\zeta'})$ for all $\zeta' \in \{\zeta^a \colon a \in \mathcal{K}\}$. Applying the inverse isomorphism χ of χ', we have

$$\chi\left(\sum_{a\in\mathcal{K}} b(x^a \bmod \mu_1), \ldots, \sum_{a\in\mathcal{K}} b(x^a \bmod \mu_d)\right) = \sum_{a\in\mathcal{K}} b(x^a \bmod \Phi_r)$$

a preimage of α in $\mathcal{R}$. Finally, let $\mu_1, \ldots, \mu_d$ be the d irreducible factors of Φ_{p^e} and $\varphi_{\zeta_1}, \ldots, \varphi_{\zeta_d}$ the corresponding canonical isomorphisms with $\zeta_i \in \{\zeta^a \colon a \in \mathcal{K}\}$ and $\mu_i(\zeta_i) = 0$ for $1 \le i \le d$. We define the map

$$\varphi \colon \begin{array}{rcl} \mathbb{F}_q(\alpha) & \to & \mathcal{R} = \mathbb{F}_q[x]/(\Phi_r) \\ A & \mapsto & \chi(\varphi_{\zeta_1}(A), \ldots, \varphi_{\zeta_d}(A)). \end{array} \tag{7.20}$$

which is a homomorphism of rings. If A is given as a linear combination of the conjugates of α then

$$\varphi\Big(\sum_{0\le h<n} A_h\alpha^{q^h}\Big) = \sum_{0\le i<n} A_i \sum_{a\in\mathcal{K}} b(x^a \bmod \Phi_r).$$

Summarizing the discussion above, we have principally found a way to exchange normal basis representation and a representation in a larger polynomial ring. We will see in Section 8 that the choice of the ring homomorphism φ enables us to do this change fast in both directions. Thus, φ opens the door to replace multiplication in $\mathbb{F}_{q^n}$ given with respect to a normal basis $\mathcal{N}$ by polynomial multiplication in $\mathcal{R}$.

7.4. Field towers, traces, and normal elements. We conclude this section by collecting some well-known properties on normal elements that are useful subsequently. The properties listed below are true not only for normal Gauß periods but for all normal bases. We will discuss the algorithmic aspects for normal bases generated by Gauß periods in the subsequent sections.

The product of normal elements. It is a well-known fact (see e.g. Semaev (1989), Lemma 1.1) that normality is inherited along a *tower of fields*

$$\mathbb{F}_q \subseteq \mathbb{F}_{q^{n_1}} \subseteq \mathbb{F}_{q^{n_1 n_2}} \subseteq \cdots \subseteq \mathbb{F}_{q^{n_1 \cdots n_t}},$$

whenever the degrees $n_1, \ldots, n_t \geq 1$ are pairwise co-prime.

FACT 7.21 (Semaev 1989). *Let n_1 and n_2 be two co-prime positive integers, $n = n_1 \cdot n_2$, and α_i be a normal element in $\mathbb{F}_{q^{n_i}}$ over $\mathbb{F}_q$ for $i = 1, 2$. Then $\alpha = \alpha_1 \cdot \alpha_2$ is normal in $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$.*

PROOF. Since n_1 and n_2 are co-prime, the field $\mathbb{F}_{q^n} = \mathbb{F}_{q^{n_1}}(\mathbb{F}_{q^{n_2}}) = \mathbb{F}_{q^{n_2}}(\mathbb{F}_{q^{n_1}})$ is the smallest extension field that contains both $\mathbb{F}_{q^{n_1}}$ and $\mathbb{F}_{q^{n_2}}$. Hence, the set $\mathcal{N} = \{\alpha_1^{q^{h_1}} \cdot \alpha_2^{q^{h_2}} : 0 \leq h_1 < n_1 \text{ and } 0 \leq h_2 < n_2\}$ is a basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$: By construction, any element in $\mathbb{F}_{q^n} = \mathbb{F}_{q^{n_1}}(\mathbb{F}_{q^{n_2}})$ can be written as a finite linear combination of elements of $\mathbb{F}_{q^{n_2}}$ with coefficients in $\mathbb{F}_{q^{n_1}}$. But $\mathcal{N}_i = (\alpha_i, \ldots, \alpha_i^{q^{n_i - 1}})$ is a basis for $\mathbb{F}_{q^{n_i}}$ for $i = 1, 2$, and thus any element of $\mathbb{F}_{q^n}$ is generated by a linear combination of the set $\mathcal{N}$. Since $\#\mathcal{N} = n_1 \cdot n_2 = n$, it is indeed a basis. By assumption we have $\gcd(n_1, n_2) = 1$, and by the Chinese Remainder Theorem there exist $u_1, u_2 \in \mathbb{Z}$ such that $u_i n_i = 1 \bmod n_{3-i}$ and $h = h_1 \cdot u_2 n_2 + h_2 \cdot u_1 n_1$ where $h_i = h \bmod n_i$ for $i = 1, 2$. But $\alpha_i^{q^{n_i}} = \alpha_i$ for $i = 1, 2$, which proves $\alpha^{q^h} = (\alpha_1 \cdot \alpha_2)^{q^{h_1 u_2 n_2 + h_2 u_1 n_1}} = \alpha_1^{q^{h_1}} \cdot \alpha_2^{q^{h_2}}$. The map $h \mapsto (h_1 \bmod n_1, h_2 \bmod n_2)$ is bijective. Thus, $\mathcal{N} = \{\alpha^{q^h} : 0 \leq h < n\}$ and the claim follows. □

Fact 7.21 shows a way to compute the multiplication matrix $T_{\mathcal{N}}$ of the normal basis $\mathcal{N} = (\alpha, \ldots, \alpha^{q^{n_1 n_2 - 1}})$ if $\gcd(n_1, n_2) = 1$ and the matrices $T_{\mathcal{N}_i}$ are already given for $i = 1, 2$. This was also done inSemaev (1989), Lemma 1.1.

FACT 7.22 (Semaev 1989). *Let n_1, n_2 and α_1, α_2 as in Fact 7.21 and set $n = n_1 \cdot n_2$. Let $T_{\mathcal{N}_1} = (u_{j_1,h_1})_{0 \leq j_1, h_1 < n_1}$ and $T_{\mathcal{N}_2} = (v_{j_2,h_2})_{0 \leq j_2, h_2 < n_2}$ be the multiplication matrices of $\mathcal{N}_i = \{\alpha_i^{q^h} : 0 \leq h < n\}$ for $i = 1, 2$.*

(i) *The multiplication matrix* $T_{\mathcal{N}} = (t_{j,h})_{0 \le j,h < n}$ *of* $\alpha = \alpha_1 \cdot \alpha_2$ *is given by*

$$t_{j,h} = u_{j_1,h_1} \cdot v_{j_2,h_2}$$

where $j \equiv j_i \bmod n_i$ *and* $h \equiv h_i \bmod n_i$ *for* $i = 1, 2$.

(ii) *The density* $d_{\mathcal{N}}$ *of* $T_{\mathcal{N}}$ *is the product of the densities* $d_{\mathcal{N}_1}$ *and* $d_{\mathcal{N}_2}$ *of* $T_{\mathcal{N}_1}$ *and* $T_{\mathcal{N}_2}$, *respectively.*

(iii) *The multiplication matrix* $T_{\mathcal{N}}$ *can be calculated with* $d_{\mathcal{N}} = d_{\mathcal{N}_1} \cdot d_{\mathcal{N}_2}$ *multiplications in* $\mathbb{F}_q$ *from* $T_{\mathcal{N}_1}$ *and* $T_{\mathcal{N}_2}$.

PROOF. By definition of $T_{\mathcal{N}} = (t_{j,h})_{0 \le j,h < n}$ we have $\alpha^{q^j} \cdot \alpha = \sum_{0 \le h < n} t_{j,h} \alpha^{q^h}$ for $0 \le j < n$. Fact 7.21 gives

$$\begin{aligned}
\alpha^{q^j} \cdot \alpha &= (\alpha_1 \cdot \alpha_2)^{q^j} \cdot (\alpha_1 \cdot \alpha_2) = \left(\alpha_1^{q^j} \cdot \alpha_1\right) \cdot \left(\alpha_2^{q^j} \cdot \alpha_2\right) \\
&= \left(\alpha_1^{q^{j_1}} \cdot \alpha_1\right) \cdot \left(\alpha_2^{q^{j_2}} \cdot \alpha_2\right) = \left(\sum_{0 \le h_1 < n_1} u_{j_1,h_1} \alpha_1^{q^{h_1}}\right) \cdot \left(\sum_{0 \le h_2 < n_2} v_{j_2,h_2} \alpha_2^{q^{h_2}}\right) \\
&= \sum_{0 \le h_1 < n_1} \sum_{0 \le h_2 < n_2} u_{j_1,h_1} \cdot v_{j_2,h_2} \left(\alpha_1^{q^{h_1}} \cdot \alpha_2^{q^{h_2}}\right) = \sum_{0 \le h < n} (u_{j_1,h_1} \cdot v_{j_2,h_2}) \alpha^{q^h}
\end{aligned}$$

where $h_i \equiv h \bmod n_i$ and $j_i \equiv j \bmod n_i$ for $i = 1, 2$. A comparison of the coefficients proves (i). An entry $t_{j,h} = u_{j_1,h_1} \cdot v_{j_2,h_2}$ in $T_{\mathcal{N}}$ is non-zero if and only if both entries in $T_{\mathcal{N}_1}$ and $T_{\mathcal{N}_2}$ are non-zero. Therefore, we have exactly $d_{\mathcal{N}_1} \cdot d_{\mathcal{N}_2}$ non-zero entries in $T_{\mathcal{N}}$ as claimed in (ii). We observe that a non-zero entry can be calculated with one multiplication in $\mathbb{F}_q$, hence, (iii) follows immediately. □

The trace of a normal element. Another basic tool which inherits normality is given by the trace map defined in (6.9). We recall some properties of the trace, see e.g. Lidl & Niederreiter (1983), Theorem 2.23(iii)-(v) and Theorem 2.26, respectively, for a proof.

FACT 7.23. *Let* m *and* n *be positive integers and* $\mathbb{F}_q$ *be the finite field with* q *elements.*

(i) *The trace* $\mathrm{Tr}_{q^n/q}$ *is a linear transformation from* $\mathbb{F}_{q^n}$ *onto* $\mathbb{F}_q$.

(ii) $\mathrm{Tr}_{q^n/q}(A) = nA$ *for* $A \in \mathbb{F}_q$, *and* $\mathrm{Tr}_{q^n/q}(A^q) = \mathrm{Tr}_{q^n/q}(A)$ *for* $A \in \mathbb{F}_{q^n}$.

(iii) *The trace is transitive, i.e.* $\mathrm{Tr}_{q^{mn}/q}(A) = \mathrm{Tr}_{q^n/q}(\mathrm{Tr}_{q^{mn}/q^n}(A))$ *for all* $A \in \mathbb{F}_{q^{mn}}$.

The next remark is also true for all Galois extensions over a finite field, see Hachenberger (1997), Lemma 5.3. Informally spoken, the trace map inherits normality downwards a field tower. We have already stated that multiplication induces normality upwards.

REMARK 7.24. *Let n_1 and n_2 be two co-prime positive integers and $n = n_1 \cdot n_2$. If α is normal in $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ then $\mathrm{Tr}_{q^n/q^{n_1}}(\alpha)$ is normal in $\mathbb{F}_{q^{n_1}}$ over $\mathbb{F}_q$.*

PROOF. Set $\alpha_1 = \mathrm{Tr}_{q^n/q^{n_1}}(\alpha)$. We have to show that $\alpha_1, \ldots, \alpha_1^{q^{n_1-1}}$ are linearly independent over $\mathbb{F}_q$. Let $A_0, \ldots, A_{n_1-1} \in \mathbb{F}_q$ such that $\sum_{0 \le h < n_1} A_h \alpha_1^{q^h} = 0$. Then

$$\begin{aligned} 0 &= \sum_{0 \le h_1 < n_1} A_{h_1} \alpha_1^{q^{h_1}} = \sum_{0 \le h_1 < n_1} A_{h_1} \mathrm{Tr}_{q^n/q^{n_1}}(\alpha)^{q^{h_1}} \\ &= \sum_{0 \le h_1 < n_1} A_{h_1} \left(\sum_{0 \le i < n/n_1} \alpha^{q^{in_1}} \right)^{q^{h_1}} = \sum_{0 \le i < n/n_1} \sum_{0 \le h_1 < n_1} A_{h_1} \alpha^{q^{in_1+h_1}} \\ &= \sum_{0 \le h < n} A_{h \bmod n_1} \alpha^{q^h}. \end{aligned}$$

But α is normal in $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$, and hence, all $A_0, \ldots, A_{n_1-1}$ are zero. □

In the special case where $n = n_1 \cdot n_2$ is the product of two co-prime factors we get some further useful properties.[53]

LEMMA 7.25. *Let n_1 and n_2 be co-prime positive integers, $n = n_1 \cdot n_2$, and let α_1 and α_2 be normal in $\mathbb{F}_{q^{n_1}}$ and $\mathbb{F}_{q^{n_2}}$ over $\mathbb{F}_q$, respectively. Then*

(i) $\mathrm{Tr}_{q^n/q^{n_2}}(\alpha_1 \cdot \alpha_2) = \mathrm{Tr}_{q^{n_1}/q}(\alpha_1) \cdot \alpha_2$ *and*

(ii) α_2 *is normal in* $\mathbb{F}_{q^n}$ *over* $\mathbb{F}_{q^{n_1}}$.

PROOF. (i) We have

$$\begin{aligned} \mathrm{Tr}_{q^n/q^{n_2}}(\alpha_1 \cdot \alpha_2) &= \sum_{0 \le i < n/n_2} (\alpha_1 \cdot \alpha_2)^{q^{in_2}} \\ &= \sum_{0 \le i < n/n_2} \alpha_1^{q^{in_2}} \cdot \alpha_2^{q^{in_2}} = \alpha_2 \cdot \sum_{0 \le i < n/n_2} \alpha_1^{q^{in_2}} \end{aligned}$$

since $\alpha_2 \in \mathbb{F}_{q^{n_2}}$, i.e. $\alpha_2^{q^{in_2}} = \alpha_2$ for all $1 \le i < \frac{n}{n_2}$. Moreover, the map $\psi_{n_2} \colon \{0, \ldots, n_1-1\} \to \{0, \ldots, n_1-1\}$ with $\psi_{n_2}(i) = n_2 i \operatorname{rem} n_1$ is a bijection and hence

$$\sum_{0 \le i < n/n_2} \alpha_1^{q^{in_2}} = \sum_{0 \le i < n_1} \alpha_1^{q^i} = \mathrm{Tr}_{q^{n_1}/q}(\alpha_1).$$

[53] A proof of Lemma 7.25(i) is also given in Jungnickel (1993), Lemma 5.1.8. A special version of Lemma 7.25(ii) is cited in Agnew *et al.* (1993) for optimal normal bases.

(ii) Since $\mathcal{N}_2 = (\alpha_2, \ldots, \alpha_2^{q^{n_2-1}})$ is a basis for $\mathbb{F}_{q^{n_2}}$, the set $\mathcal{N}_2$ is a basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_{q^{n_1}}$. By assumption, n_1 and n_2 are co-prime, and hence, the map $\psi_{n_1} : \{0, \ldots, n_2 - 1\} \to \{0, \ldots, n_2 - 1\}$ with $\psi_{n_1}(i) = n_1 i \operatorname{rem} n_2$ is a bijection. Therefore, the set $\{\alpha_2^{q^{n_1 h}} : 0 \leq h < n_2\} = \{\alpha_2^{q^h} : 0 \leq h < n_2\}$ is the set of all n_2 conjugates of α_2 over $\mathbb{F}_{q^{n_1}}$, and $\mathcal{N}_2$ is a normal basis over $\mathbb{F}_{q^{n_1}}$ as claimed. □

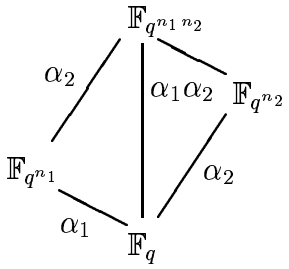


Figure 7.2: A tower of fields given by normal elements if $\gcd(n_1, n_2) = 1$.

8. The prime power case

We are now ready to develop an algorithm that integrates polynomial multiplication in a normal basis representation whenever the normal element is a Gauß period. In this section, we restrict to the case where $\alpha = \sum_{a\in\mathcal{K}} \sum_{0\le s<e} \zeta^{ap^s}$ is a prime or prime power Gauß period of type $(n,\mathcal{K})$ over $\mathbb{F}_q$, i.e. $r = p^e$. The main result of this section generalizes the approach that was described in Gao *et al.* (1995) and Gao *et al.* (2000) for prime Gauß periods.

RESULT 8.1. *Let p be a prime, e be a positive integer, and α be a normal prime power Gauß period of type $(n,\mathcal{K})$ over $\mathbb{F}_q$, where $\mathcal{K}$ is a subgroup of $\mathbb{Z}_{p^e}^\times$. Two elements of $\mathbb{F}_{q^n}$ expressed in the normal basis $\mathcal{N} = (\alpha,\dots,\alpha^{q^{n-1}})$ can be multiplied with at most $O(p^e \log p^e \cdot \operatorname{loglog} p^e)$ operations in $\mathbb{F}_q$.*

The underlying algorithm is the second corner pillar of this thesis. The algorithm consists of three parts: multiplication in $\mathbb{F}_q[x]/(x^{p^e}-1)$, sorting the product to identify prime (power) Gauß periods in subfields of $\mathbb{F}_{q^n}$, and then applying the trace map to return to the linear combination of the conjugates of the prime (power) Gauß period. We discuss this in detail in Section 8.1. In doing so, we get tools to derive some further results. We may perform the Extended Euclidean Algorithm 5.29 to compute the inverse in the case of normal prime power Gauß periods (Section 8.2). The algorithmic tools are also used to generalize an algorithm of Wassermann (1990); we calculate the multiplication matrix of normal prime power Gauß periods (Section 8.3). With the help of these tools we prove the Normal Gauß period theorem 7.5 for the special case of prime (power) Gauß periods (Section 8.4).

8.1. An algorithm for fast multiplication. We start with an example that will illustrate our algorithmic ideas.

EXAMPLE 8.2. Let ζ be a primitive 9-th root of unity, and let α be the normal Gauß period of type $(3,\{1,8\})$ over $\mathbb{F}_2$ as in Example 7.4(ii). The conjugates of $\alpha = \zeta+\zeta^3+\zeta^8+\zeta^6$ are $\alpha^{2^1} = \zeta^2+\zeta^6+\zeta^7+\zeta^3$ and $\alpha^{2^2} = \zeta^4+\zeta^3+\zeta^5+\zeta^6$.

(i) To calculate the product $\alpha^{2^2}\cdot\alpha$ as linear combination of α,α^2,α^4, we regard the conjugates of α as elements of $\mathbb{F}_2(\zeta)$. The product in this extension field is

$$\begin{aligned}\alpha^4\cdot\alpha &= (\zeta^4+\zeta^3+\zeta^5+\zeta^6)\cdot(\zeta+\zeta^3+\zeta^8+\zeta^6)\\ &= \zeta^5+\zeta^{10}+\zeta^4+\zeta^8+\zeta^{13}+\zeta^{14}\end{aligned}$$

Since ζ is a primitive 9-th root of unity over $\mathbb{F}_2$, we have $\zeta^9 = 1$ and $\alpha^4\cdot\alpha = \zeta+\zeta^8$. The summands are summands of α. We complete the

missing terms to get

$$\alpha^4 \cdot \alpha = (\zeta + \zeta^3 + \zeta^8 + \zeta^6) + \zeta^3 + \zeta^6.$$

(ii) Observe that ζ^3 and ζ^6 are primitive 3-rd roots of unity over $\mathbb{F}_2$. We apply the canonical projection $\pi_3 \colon \mathbb{Z}_9^\times \to \mathbb{Z}_3^\times$ as defined in (7.8). Then $\pi_3(\{1,8\}) = \{1,2\} = \mathbb{Z}_3^\times$ and hence $n' = \frac{\phi(3)}{\#\{1,2\}} = 1$. Thus, the projection generates the prime Gauß period $\alpha_3 = (\zeta^3) + (\zeta^3)^2$ over $\mathbb{F}_2$. We substitute $\zeta^3 + \zeta^6$ by α_3 to get

$$\alpha^4 \cdot \alpha = \alpha + \alpha_3.$$

(iii) In order to express α_3 as a linear combination of the conjugates of α we compute the trace of α over $\mathbb{F}_{2^1}$:

$$\begin{aligned} \mathrm{Tr}_{2^3/2^1}(\alpha) &= \sum_{0 \le i < 3} \alpha^{2^i} = \alpha + \alpha^2 + \alpha^4 \\ &= (\zeta + \zeta^3 + \zeta^8 + \zeta^6) + (\zeta^2 + \zeta^6 + \zeta^7 + \zeta^3) + (\zeta^4 + \zeta^3 + \zeta^5 + \zeta^6) \\ &= \zeta + \zeta^7 + \zeta^4 + \zeta^8 + \zeta^2 + \zeta^5 + \zeta^3 + \zeta^6. \end{aligned}$$

We resort the summands and apply (7.15), i.e. $0 = \Phi_3(\zeta^3) = 1 + \zeta^3 + \zeta^6$ to get

$$\begin{aligned} \mathrm{Tr}_{2^3/2^1}(\alpha) &= \zeta \cdot (1 + \zeta^6 + \zeta^3) + \zeta^2 \cdot (\zeta^6 + 1 + \zeta^3) + \zeta^3 + \zeta^6 \\ &= \zeta^3 + \zeta^6 = \alpha_3. \end{aligned}$$

Indeed, the trace describes a linear combination of the conjugates of α for α_3. We insert this linear combination

$$\alpha^4 \cdot \alpha = \alpha + \alpha_3 = \alpha + \mathrm{Tr}_{2^3/2^1}(\alpha) = \alpha^2 + \alpha^4$$

which completes the computation. ◊

We will show that the map $\varphi \colon \mathbb{F}_q(\alpha) \to \mathcal{R} = \mathbb{F}_q[x]/(\Phi_{p^e})$ as in (7.20) is in fact an injective ring homomorphism if α is normal over $\mathbb{F}_q$.

A sum of Gauß periods. The door to our algorithm tool box opens by a careful look at the summands of the product $\varphi(A) \cdot \varphi(B)$. We claim that a preimage of φ of this product in $\mathbb{F}_q[x]/(x^{p^e} - 1)$ can be written in a particular way. We note that $x^a \equiv x^b \bmod (x^{p^e} - 1)$ if $a \equiv b \bmod p^e$. The following idea also carries the key to the computation of the multiplication matrix $T_{\mathcal{N}}$, see Section 8.3.

For all $0 \le i < n$, we define the positive integers

$$\begin{aligned} u_{\ell,h}^{(i)} &= \#\{a \in \mathcal{K} \colon 1 + aq^i \in p^{e-\ell} q^h \mathcal{K}\} \quad \text{for } 0 < \ell \le e \text{ and } 0 \le h < n_\ell, \\ v_{\ell,h}^{(i)} &= \#\{a \in \mathcal{K} \colon 1 + ap^\ell q^i \in q^h \mathcal{K}\} \quad \text{for } 0 < \ell < e \text{ and } 0 \le h < n_\ell. \end{aligned} \tag{8.3}$$

Furthermore, we set

$$u_{0,0}^{(i)} = \begin{cases} 1 & \text{if there is } a \in q^i\mathcal{K} \text{ such that } 1 + aq^i \equiv 0 \bmod p^e, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

These numbers define the *special form* of the preimage in $\mathbb{F}_q[x]/(x^{p^e} - 1)$ of $\varphi(A) \cdot \varphi(B)$, we are looking for. Subsequently, we suppose that $\langle q, \mathcal{K}\rangle = \mathbb{Z}_{p^e}^{\times}$. We fix some further notation.

NOTATION 8.4. *Let ζ be a primitive p^e-th root of unity. For $0 < \ell \leq e$ let π_{p^ℓ} be the canonical projection from $\mathbb{Z}_{p^e}^{\times}$ onto $\mathbb{Z}_{p^\ell}^{\times}$. Set $k_\ell = \#\pi_{p^\ell}(\mathcal{K})$ and $n_\ell = \frac{\phi(p^\ell)}{k_\ell}$. The Gauß period of type $(n_\ell, \pi_{p^\ell}(\mathcal{K}))$ over $\mathbb{F}_q$ with respect to $\zeta_\ell = \zeta^{p^{e-\ell}}$ is denoted by α_ℓ. We set $n_0 = k_0 = 1$.*

Since φ is additive, it is sufficient to look at the following product.

LEMMA 8.5. *Let $0 \leq i < n$ and $\mathbb{F}$ be the prime subfield of $\mathbb{F}_q$. Then there are $C_0^{(i)}$ and $C_{\ell,h}^{(i)}$ in $\mathbb{F}$ for $0 < \ell \leq e$ and $0 \leq h < n_\ell$ such that*

$$\begin{aligned} &\left(\sum_{a\in\mathcal{K}} \sum_{0\leq s<e} x^{ap^s q^i}\right) \cdot \left(\sum_{b\in\mathcal{K}} \sum_{0\leq s'<e} x^{bp^{s'}}\right) \\ \equiv\ & C_0^{(i)} + \sum_{0<\ell\leq e} \sum_{0\leq h<n_\ell} C_{\ell,h}^{(i)} \left(\sum_{a\in\pi_{p^\ell}(\mathcal{K})} \sum_{0\leq s<\ell} (x^{p^{e-\ell}})^{ap^s}\right)^{q^h} \bmod (x^{p^e} - 1). \end{aligned}$$

Since ζ is a root of $(x^{p^e} - 1)$, the product of α^{q^i} times α can be written as a sum of those Gauß periods α_ℓ which are given by the canonical projection of $\mathcal{K}$ onto $\mathbb{Z}_{p^\ell}^{\times}$.

COROLLARY 8.6. *Let α be the Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ with respect to ζ. For $0 < \ell \leq e$, let α_ℓ be the Gauß period of type $(n_\ell, \pi_{p^\ell}(\mathcal{K}))$ over $\mathbb{F}_q$ with respect to $\zeta^{p^{e-\ell}}$. For $0 \leq i < n$, let $C_0^{(i)}$ and $C_{\ell,h}^{(i)}$ for $0 \leq \ell < e$ and $0 \leq h < n_\ell$ as in Lemma 8.5. Then*

$$\alpha^{q^i} \cdot \alpha \ = \ C_0^{(i)} + \sum_{0<\ell\leq e} \sum_{0\leq h<n_\ell} C_{\ell,h}^{(i)} \alpha_\ell^{q^h}.$$

We start with a proposition that describes the coefficients of the preimage of $\varphi(A) \cdot \varphi(B)$ in $\mathbb{F}_q[x]/(x^{p^e} - 1)$ in terms of $u_{\ell,h}^{(i)}$ and $v_{\ell,h}^{(i)}$.

PROPOSITION 8.7. *Let $0 \le i < n$ be fixed and $u^{(i)}_{\ell,h}$ and $v^{(i)}_{\ell,h}$ as in (8.3). Set*

$$\begin{aligned} C'_0 &= k \cdot \textstyle\sum_{0\le\ell\le e} \Big((e-\ell)\cdot \sum_{0\le h<n_\ell} u^{(i)}_{\ell,h}\Big) && \text{and} \\ C'_{p^{e-\ell}q^h} &= \tfrac{k}{k_\ell}\cdot\Big(\textstyle\sum_{\ell\le s\le e} u^{(i)}_{s,h} + \sum_{0<s<\ell}(v^{(i)}_{s,h} + v^{(n-i)}_{s,h-i})\Big) \\ & && \text{for all } 0<\ell\le e \text{ and } 0\le h<n_\ell. \end{aligned}$$

Then

$$\begin{aligned} &\Big(\sum_{a\in\mathcal{K}}\sum_{0\le s<e} x^{ap^sq^i}\Big)\cdot\Big(\sum_{b\in\mathcal{K}}\sum_{0\le s'<e} x^{bp^{s'}}\Big) \\ \equiv\ & C'_0 + \sum_{0<\ell\le e}\sum_{0\le h<n_\ell} C'_{p^{e-\ell}q^h} \sum_{a\in\pi_{p^\ell}(\mathcal{K})} (x^{p^{e-\ell}})^{aq^h} \bmod (x^{p^e}-1). \end{aligned}$$

PROOF. A straightforward computation gives

$$\begin{aligned} &\Big(\sum_{a\in\mathcal{K}}\sum_{0\le s<e} x^{ap^sq^i}\Big)\cdot\Big(\sum_{b\in\mathcal{K}}\sum_{0\le s'<e} x^{bp^{s'}}\Big) = \sum_{a,b\in\mathcal{K}}\sum_{0\le s,s'<e} x^{ap^sq^i+bp^{s'}} \\ =\ & \sum_{a,b\in\mathcal{K}}\Big(\sum_{0\le s<e} x^{ap^sq^i+bp^{s+0}} + \sum_{0<\ell<e}\sum_{0\le s<e-\ell}\Big(x^{ap^{s+\ell}q^i+bp^s} + x^{ap^sq^i+bp^{s+\ell}}\Big)\Big) \\ \equiv\ & \sum_{a\in\mathcal{K}}\sum_{b\in\mathcal{K}}\sum_{0\le s<e} x^{bp^s(1+aq^i)} + \sum_{a\in\mathcal{K}}\sum_{b\in\mathcal{K}}\sum_{0<\ell<e}\sum_{0\le s<e-\ell} x^{bp^s(1+ap^\ell q^i)} \\ & + \sum_{b\in\mathcal{K}}\sum_{a\in\mathcal{K}}\sum_{0<\ell<e}\sum_{0\le s<e-\ell} x^{ap^sq^i(1+bp^\ell q^{n-i})} \bmod (x^{p^e}-1). \end{aligned}$$

We consider the three major summands separately.

Fix $a \in \mathcal{K}$. Then $1+aq^i$ is either equal 0 modulo p^e or there are $0<\ell\le e$ and $0\le h<n_\ell$ such that $1+aq^i \in p^{e-\ell}q^h\mathcal{K} \subseteq \mathbb{Z}_{p^e}$. Then

$$\sum_{b\in\mathcal{K}}\sum_{0\le s<e} x^{bp^s(1+aq^i)} \equiv \sum_{b\in\mathcal{K}}\sum_{0\le s<e} x^0 \equiv ke \bmod (x^{p^e}-1)$$

if $1+aq^i \equiv 0 \bmod p^e$, and otherwise we have

$$\begin{aligned} &\sum_{b\in\mathcal{K}}\sum_{0\le s<e} x^{bp^s(1+aq^i)} \equiv \sum_{b\in\mathcal{K}}\sum_{0\le s<e} x^{bp^sp^{e-\ell}q^h} \\ \equiv\ & \sum_{b\in\mathcal{K}}\Big(\sum_{0\le s<\ell} x^{bp^{e-(\ell-s)}q^h} + \sum_{\ell\le s<e} x^{bp^{e+(s-\ell)}q^h}\Big) \\ \equiv\ & \sum_{b\in\mathcal{K}}\sum_{0\le s<\ell} x^{bp^{e-(\ell-s)}q^h} + \sum_{b\in\mathcal{K}}(e-\ell) \\ \equiv\ & \sum_{0<s\le\ell}\frac{k}{k_s}\sum_{b\in\pi_{p^s}(\mathcal{K})} (x^{p^{e-s}})^{bq^h} + k(e-\ell) \bmod (x^{p^e}-1). \end{aligned}$$

If a runs through $\mathcal{K}$ then we get the first intermediate result as

$$\begin{aligned}
&\sum_{a\in\mathcal{K}}\left(\sum_{b\in\mathcal{K}}\sum_{0\le s<e} x^{bp^s(1+aq^i)}\right)\\
\equiv\ &\sum_{0<\ell\le e}\sum_{0\le h<n_\ell} u^{(i)}_{\ell,h}\cdot\left(\sum_{0<s\le\ell}\frac{k}{k_s}\sum_{b\in\pi_{p^s}(\mathcal{K})}(x^{p^{e-s}})^{bq^h}+k(e-\ell)\right)+u^{(i)}_{0,0}ke\\
\equiv\ &k\cdot\sum_{0\le\ell<e}\left((e-\ell)\cdot\sum_{0\le h<n_\ell}u^{(i)}_{\ell,h}\right)\\
&+\sum_{0<\ell\le e}\sum_{0\le h<n_\ell}\left(\frac{k}{k_\ell}\sum_{\ell\le s\le e}u^{(i)}_{s,h}\right)\left(\sum_{b\in\pi_{p^\ell}(\mathcal{K})}(x^{p^{e-\ell}})^b\right)^{q^h}\bmod(x^{p^e}-1).
\end{aligned}$$

For the second sum, we fix $a\in\mathcal{K}$ and $0<\ell<e$. Since $ap^\ell q^i$ is divisible by p, we have $\gcd(1+ap^\ell q^i,p^e)=1$, i.e. $1+ap^\ell q^h\in\mathbb{Z}^\times_{p^e}$. By assumption, we have $\langle q,\mathcal{K}\rangle=\mathbb{Z}^\times_{p^e}$, and there is $0\le h<n$ such that $1+ap^\ell q^i\in q^h\mathcal{K}$. Then we get

$$\begin{aligned}
&\sum_{b\in\mathcal{K}}\sum_{0\le s<e-\ell}x^{bp^s(1+ap^\ell q^i)}\equiv\sum_{b\in\mathcal{K}}\sum_{0\le s<e-\ell}x^{bp^sq^h}\\
\equiv&\sum_{b\in\mathcal{K}}\sum_{\ell<s\le e}(x^{p^{e-s}})^{bq^h}\equiv\sum_{\ell<s\le e}\frac{k}{k_s}\sum_{b\in\pi_{p^s}(\mathcal{K})}(x^{p^{e-s}})^{bq^h}\bmod(x^{p^e}-1).
\end{aligned}$$

If a runs through $\mathcal{K}$ then the sum over all $0<\ell<e$ is given by

$$\begin{aligned}
&\sum_{0<\ell<e}\sum_{a\in\mathcal{K}}\left(\sum_{b\in\mathcal{K}}\sum_{0\le s<e-\ell}x^{bp^s(1+ap^\ell q^i)}\right)\\
\equiv&\sum_{0<\ell<e}\sum_{0\le h<n_\ell}v^{(i)}_{\ell,h}\cdot\left(\sum_{\ell<s\le e}\frac{k}{k_s}\cdot\sum_{b\in\pi_{p^s}(\mathcal{K})}(x^{p^{e-s}})^{bq^h}\right)\\
\equiv&\sum_{1<\ell\le e}\sum_{0\le h<n_\ell}\left(\frac{k}{k_\ell}\sum_{0<s<\ell}v^{(i)}_{s,h}\right)\left(\sum_{b\in\pi_{p^\ell}(\mathcal{K})}(x^{p^{e-\ell}})^b\right)^{q^h}.
\end{aligned}$$

By changing the rôles of a and b and substituting i by $n-i$, we get the formula

for the third summand:

$$\begin{aligned}
&\sum_{0<\ell<e}\sum_{b\in\mathcal{K}}\left(\sum_{a\in\mathcal{K}}\sum_{0\le s<e-\ell} x^{ap^s(1+bp^\ell q^{n-i})}\right)^{q^i}\\
&\equiv \sum_{1<\ell\le e}\sum_{0\le h<n_\ell}\left(\frac{k}{k_\ell}\sum_{0<s<\ell} v_{s,h}^{(n-i)}\right)\left(\sum_{a\in\pi_{p^\ell}(\mathcal{K})}(x^{p^{e-\ell}})^a\right)^{q^{i+h}}\\
&\equiv \sum_{1<\ell\le e}\sum_{0\le h<n_\ell}\left(\frac{k}{k_\ell}\sum_{0<s<\ell} v_{s,h-i}^{(n-i)}\right)\left(\sum_{a\in\pi_{p^\ell}(\mathcal{K})}(x^{p^{e-\ell}})^a\right)^{q^{h}} \mod (x^{p^e}-1).
\end{aligned}$$

This completes the proof. □

With the help of this first proposition, we can group all summands of the preimage of $\varphi(A)\cdot\varphi(B)$ in $\mathbb{F}_q[x]/(x^{p^e}-1)$—except the absolute coefficient—in terms $\sum_{a\in\pi_{p^\ell}(\mathcal{K})}(x^{p^{e-\ell}})^{aq^h}$ with $0<\ell<e$ and $0\le h<n_\ell$. Let $0\le i<n$ be fixed as before; we omit it in the notation. Now our approach is to resort these terms into sums which are preimages of α_ℓ, for $0<\ell\le e$, in $\mathcal{R}$. This is obvious but a little bit technical. Thus, we want to define two useful sequences of integers for all $0<\ell\le e$ and $0\le h<n_\ell$:

$$\begin{array}{lcll}
D_{\ell,h}^{(e)} & = & 0 & \text{and}\\
C_{\ell,h} & = & C'_{p^{e-\ell}q^h} - D_{\ell,h}^{(\ell)} & \text{and}\\
D_{\ell,h}^{(s)} & = & D_{\ell,h}^{(s+1)} + \frac{k_{s+1}}{k_\ell}\sum_{0\le j<\frac{n_{s+1}}{n_\ell}} C_{s+1,h+jn_\ell} & \text{for } \ell\le s<e.
\end{array} \tag{8.8}$$

Informally spoken, the $D_{\ell,h}^{(s)}$ are those parts of the $C'_{p^{e-\ell}q^h}$ which have already been identified as Gauß periods. We give some alternative computations of the $D_{\ell,h}^{(s)}$ to illustrate this.

REMARK 8.9. *Let $D_{\ell,h}^{(s)}$ and $C_{\ell,h}$ as above. Then*

(i) $D_{\ell,h}^{(s)} = \sum_{s\le s'<e}\frac{k_{s'+1}}{k_\ell}\left(\sum_{0\le j<\frac{n_{s'+1}}{n_\ell}} C_{s'+1,h+jn_\ell}\right)$, *for* $0<\ell\le s<e$,

(ii) $D_{\ell,h}^{(\ell+1)} = \frac{k_{\ell+1}}{k_\ell}\sum_{0\le j<\frac{n_{\ell+1}}{n_\ell}} D_{\ell+1,h+jn_\ell}^{(\ell+1)}$, *for* $0<\ell<e$,

(iii) $D_{\ell,h}^{(\ell)} = \frac{k_{\ell+1}}{k_\ell}\sum_{0\le j<\frac{n_{\ell+1}}{n_\ell}}\left(D_{\ell+1,h+jn_\ell}^{(\ell+1)} + C_{\ell+1,h+jn_\ell}\right)$, *for* $0<\ell<e$.

PROOF. We prove all three formulas by induction.

(i) We prove by induction on s. For $s = e-1$, by definition we have for all $0 < \ell < e$ that

$$D_{\ell,h}^{(e-1)} = D_{\ell,h}^{(e)} + \frac{k_e}{k_\ell} \sum_{0 \le j < \frac{n_e}{n_\ell}} C_{e,h+jn_\ell}$$
$$= \sum_{e-1 \le s' < e} \frac{k_{s'+1}}{k_\ell} \left(\sum_{0 \le j < \frac{n_{s'+1}}{n_\ell}} C_{s'+1,h+jn_{e-1}} \right)$$

using $D_{\ell,h}^{(e)} = 0$. We suppose that the claimed formula is also true for $1 < s+1 < e$. Inserting the induction hypothesis into the definition of $D_{\ell,h}^{(s)}$ gives

$$D_{\ell,h}^{(s)} = D_{\ell,h}^{(s+1)} + \frac{k_{s+1}}{k_\ell} \left(\sum_{0 \le j < \frac{n_{s+1}}{n_\ell}} C_{s+1,h+jn_\ell} \right)$$
$$= \sum_{s+1 \le s' < e} \frac{k_{s'+1}}{k_\ell} \left(\sum_{0 \le j < \frac{n_{s'+1}}{n_\ell}} C_{s'+1,h+jn_\ell} \right) + \frac{k_{s+1}}{k_\ell} \sum_{0 \le j < \frac{n_{s+1}}{n_\ell}} C_{s+1,h+jn_\ell}$$
$$= \sum_{s \le s' < e} \frac{k_{s'+1}}{k_\ell} \left(\sum_{0 \le j < \frac{n_{s'+1}}{n_\ell}} C_{s'+1,h+jn_\ell} \right),$$

and the induction step is complete.

(ii) Let $0 < \ell < e$. Then

$$D_{\ell,h}^{(\ell+1)} = \sum_{\ell+1 \le s' < e} \frac{k_{s'+1}}{k_\ell} \left(\sum_{0 \le j < \frac{n_{s'+1}}{n_\ell}} C_{s'+1,h+jn_\ell} \right)$$

by (i). We resort the summands:

$$D_{\ell,h}^{(\ell+1)} = \sum_{\ell+1 \le s' < e} \frac{k_{\ell+1}}{k_\ell} \cdot \frac{k_{s'+1}}{k_{\ell+1}} \left(\sum_{0 \le j < \frac{n_{\ell+1}}{n_\ell}} \sum_{0 \le l < \frac{n_{s'+1}}{n_{\ell+1}}} C_{s'+1,h+(jn_\ell + ln_{\ell+1})} \right)$$

$$= \frac{k_{\ell+1}}{k_\ell} \cdot \sum_{0 \le j < \frac{n_{\ell+1}}{n_\ell}} \left(\sum_{\ell+1 \le s' < e} \frac{k_{s'+1}}{k_{\ell+1}} \left(\sum_{0 \le l < \frac{n_{s'+1}}{n_{\ell+1}}} C_{s'+1,(h+jn_\ell)+ln_{\ell+1}} \right) \right)$$

$$= \frac{k_{\ell+1}}{k_\ell} \cdot \sum_{0 \le j < \frac{n_{\ell+1}}{n_\ell}} D^{(\ell+1)}_{\ell+1,h+jn_\ell}.$$

Here we have used formula (i) once more.

(iii) We prove by induction on ℓ. For $\ell = e-1$, we have by the definition

$$D^{(e-1)}_{e-1,h} = D^{(e)}_{e-1,h} + \frac{k_e}{k_{e-1}} \sum_{0 \le j < \frac{n_e}{n_{e-1}}} C_{e,h+jn_{e-1}},$$

which is just the claimed formula since $D^{(e)}_{\ell,h} = 0$ for all $0 < \ell \le e$. We assume that the claim also holds for $1 < \ell + 1 < e$. Then (ii) gives

$$D^{(\ell)}_{\ell,h} = D^{(\ell+1)}_{\ell,h} + \frac{k_{\ell+1}}{k_\ell} \sum_{0 \le j < \frac{n_{\ell+1}}{k_\ell}} C_{\ell+1,h+jn_\ell}$$

$$= \frac{k_{\ell+1}}{k_\ell} \sum_{0 \le j < \frac{n_{\ell+1}}{k_\ell}} \left(D^{(\ell+1)}_{\ell+1,h+jn_\ell} + C_{\ell+1,h+jn_\ell} \right).$$

□

We prove with the help of these sequences $D^{(s)}_{\ell,h}$ and $C_{\ell,h}$ that the preimage of $\varphi(A) \cdot \varphi(B)$ in $\mathbb{F}_q[x]/(x^{p^e}-1)$ can be written as a sum of Gauß periods.

PROPOSITION 8.10. *Let $C_0 = C'_0$ and let $C_{\ell,h}$ and $D^{(s)}_{\ell,h}$ be as in (8.8). Then*

$$\left(\sum_{a \in \mathcal{K}} \sum_{0 \le s < e} x^{ap^s q^i} \right) \cdot \left(\sum_{b \in \mathcal{K}} \sum_{0 \le s' < e} x^{bp^{s'}} \right)$$

$$\equiv C_0 + \sum_{\ell' < \ell \le e} \sum_{0 \le h < n_\ell} C_{\ell,h} \cdot \left(\sum_{a \in \pi_{p^\ell}(\mathcal{K})} \sum_{0 \le s < \ell} (x^{p^{e-\ell}})^{ap^s} \right)^{q^h}$$

$$+ \sum_{0 < \ell \le \ell'} \sum_{0 \le h < n_\ell} \left(C'_{p^{e-\ell}q^h} - D^{(\ell')}_{\ell,h} \right) \cdot \left(\sum_{a \in \pi_{p^\ell}(\mathcal{K})} x^{p^{e-\ell}aq^h} \right) \bmod (x^{p^e}-1).$$

for all $0 \le \ell' \le e$.

This proposition covers Lemma 8.5 for $\ell' = 0$.

PROOF (of Proposition 8.10). By induction on $0 < \ell' \leq e$. For $\ell' = e$ the right side of the claimed equation is

$$C'_0 + 0 + \sum_{0<\ell\leq e} \sum_{0\leq h<n_\ell} \left(C'_{p^{e-\ell}q^h} - D^{(e)}_{\ell,h} \right) \cdot \left(\sum_{a\in\pi_{p^\ell}(\mathcal{K})} x^{p^{e-\ell}aq^h} \right)$$

which is just the right side of the congruence in Proposition 8.7 since $D^{(e)}_{\ell,h} = 0$ for all $0 < \ell \leq e$ and $0 \leq h < n_\ell$. Now, we suppose that the formula is also true for an $\ell \in \mathbb{N}_{>0}$ with $0 < \ell' \leq \ell \leq e$. Then for all $0 \leq h < n_{\ell'}$

$$\begin{aligned}
& \left(C'_{p^{e-\ell'}q^h} - D^{(\ell')}_{\ell',h} \right) \cdot \left(\sum_{a\in\pi_{p^{\ell'}}(\mathcal{K})} x^{p^{e-\ell'}aq^h} \right) \\
\overset{(8.8)}{\equiv}\ & C_{\ell',h} \cdot \sum_{a\in\pi_{p^{\ell'}}(\mathcal{K})} \left(\sum_{0\leq s<\ell'} x^{p^{e-\ell'}ap^sq^h} - \sum_{1\leq s<\ell'} x^{p^{e-\ell'}ap^sq^h} \right) \\
\equiv\ & \left(C_{\ell',h} \cdot \sum_{a\in\pi_{p^{\ell'}}(\mathcal{K})} \sum_{0\leq s<\ell'} x^{p^{e-\ell'}ap^sq^h} \right) - \left(C_{\ell',h} \cdot \sum_{a\in\pi_{p^{\ell'}}(\mathcal{K})} \sum_{1\leq s<\ell'} x^{p^{e-(\ell'-s)}aq^h} \right) \\
& \bmod (x^{p^e} - 1).
\end{aligned}$$

We resort the summands by adding the first term of the difference to the already collected summands

$$\begin{aligned}
& C_0 + \sum_{\ell'<\ell\leq e} \sum_{0\leq h<n_\ell} C_{\ell,h} \cdot \left(\sum_{a\in\pi_{p^\ell}(\mathcal{K})} \sum_{0\leq s<\ell} (x^{p^{e-\ell}})^{ap^s} \right)^{q^h} \\
& + \sum_{0\leq h<n_{\ell'}} C_{\ell',h} \cdot \sum_{a\in\pi_{p^{\ell'}}(\mathcal{K})} \sum_{0\leq s<\ell'} x^{p^{e-\ell'}ap^sq^h} \\
\equiv\ & C_0 + \sum_{\ell'\leq\ell\leq e} \sum_{0\leq h<n_\ell} C_{\ell,h} \cdot \left(\sum_{a\in\pi_{p^\ell}(\mathcal{K})} \sum_{0\leq s<\ell} (x^{p^{e-\ell}})^{ap^s} \right)^{q^h} \bmod (x^{p^e} - 1).
\end{aligned}$$

The remaining part is

$$\begin{aligned}
&\sum_{0<\ell<\ell'} \sum_{0\le h<n_\ell} \left(C'_{p^{e-\ell}q^h} - D^{(\ell')}_{\ell,h}\right) \cdot \left(\sum_{a\in\pi_{p^\ell}(\mathcal{K})} x^{p^{e-\ell}aq^h}\right) \\
&\quad - \sum_{0\le h<n_{\ell'}} C_{\ell',h} \cdot \sum_{a\in\pi_{p^{\ell'}}(\mathcal{K})} \sum_{1\le s<\ell'} x^{p^{e-(\ell'-s)}aq^h} \\
\equiv\ &\sum_{0<\ell<\ell'} \sum_{0\le h<n_\ell} \left(C'_{p^{e-\ell}q^h} - D^{(\ell')}_{\ell,h}\right) \cdot \left(\sum_{a\in\pi_{p^\ell}(\mathcal{K})} x^{p^{e-\ell}aq^h}\right) \\
&\quad - \sum_{0\le h<n_{\ell'}} C_{\ell',h} \cdot \sum_{1\le s<\ell'} \frac{k_{\ell'}}{k_s} \sum_{a\in\pi_{p^s}(\mathcal{K})} (x^{p^{e-s}})^{aq^h} \\
\equiv\ &\sum_{0<\ell<\ell'} \sum_{0\le h<n_\ell} \left(C'_{p^{e-\ell}q^h} - \left(D^{(\ell')}_{\ell,h} + \frac{k_{\ell'}}{k_\ell} \cdot \sum_{0\le j<\frac{n_{\ell'}}{n_\ell}} C_{\ell',h+jn_\ell}\right)\right) \\
&\quad \cdot \left(\sum_{a\in\pi_{p^\ell}(\mathcal{K})} x^{p^{e-\ell}aq^h}\right) \bmod (x^{p^e}-1).
\end{aligned}$$

But $D^{(\ell')}_{\ell,h} + \frac{k_{\ell'}}{k_\ell} \sum_{0\le i<\frac{n_{\ell'}}{n_\ell}} C_{\ell',h+in_\ell} = D^{(\ell'-1)}_{\ell,h}$ by construction in (8.8), and the induction step follows. □

Applying the trace map. The last one of our ingredients is the trace map. It points a way to write a normal Gauß period $\alpha_\ell \in \mathbb{F}_{q^{n_\ell}}$ as a linear combination of the elements of the normal basis $\mathcal{N} = (\alpha, \alpha^q, \ldots, \alpha^{q^{n-1}})$ of $\mathbb{F}_{q^n}$.

LEMMA 8.11. *Let $r = p^e$ be a prime power, and let α be a prime power Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ with respect to ζ, where $\langle q, \mathcal{K}\rangle = \mathbb{Z}^\times_{p^e}$. For any $0 < \ell \le e$, let α_ℓ be the Gauß period of type $(n_\ell, \pi_{p^\ell}(\mathcal{K}))$ over $\mathbb{F}_q$ with respect to $\zeta^{p^{e-\ell}}$. Then*

$$\sum_{0\le i<\frac{n}{n_\ell}} \alpha^{q^{in_\ell}} = p^{e-\ell}\alpha_\ell \text{ for } 0 < \ell \le e.$$

Furthermore, we have

$$\sum_{0\le i<n} \alpha^{q^i} = -p^{e-1}.$$

We again derive these formulas step-by-step, and we will give a proof of Lemma 8.11 as a conclusion at the end of this paragraph. Moreover, we show

that this lemma includes the reduction modulo Φ_{p^e} we are looking for. We start by defining a set of polynomials $\tau_0, \tau_{\ell,b} \in \mathbb{F}_q[x]$ for $0 < \ell < e$ and $b \in \pi_{p^\ell}(\mathcal{K})$. Since we are still working in the ring $\mathbb{F}_q[x]/(x^{p^e}-1)$, we assume all polynomials to be reduced modulo $x^{p^e}-1$, i.e. we identify $(a \bmod p^e) \in \mathbb{Z}_{p^e}^\times$ with its canonical representative $\bar{a} \in \mathbb{Z}$, $0 < \bar{a} < p^e$, such that $\bar{a} \equiv a \bmod p^e$. For $0 < \ell < e$ and $0 \le i < \frac{n_{\ell+1}}{n_\ell}$ and a fixed $b \in \pi_{p^\ell}(\mathcal{K})$ we define

$$\mathcal{I}_{\ell,b,i} = \{a \in \pi_{p^{\ell+1}}(\mathcal{K}) \colon a \equiv q^{-in_\ell} b \bmod p^\ell\}$$

the set of all elements in $\pi_{p^{\ell+1}}(\mathcal{K})$ that are preimages of $q^{-in_\ell}b$ under the canonical projection $\pi \colon \mathbb{Z}_{p^{\ell+1}}^\times \to \mathbb{Z}_{p^\ell}^\times$. For $0 < \ell < e$ and $b \in \pi_{p^\ell}(\mathcal{K})$, we set

$$\begin{aligned} \tau_0 &= \textstyle\sum_{0\le i<n_1} \sum_{a\in\pi_p(\mathcal{K})} (x^{p^{e-1}})^{aq^i} + 1 \in \mathbb{F}_q[x] \qquad \text{and} \\ \tau_{\ell,b} &= \textstyle\sum_{0\le i<\frac{n_{\ell+1}}{n_\ell}} \sum_{a\in\mathcal{I}_{\ell,b,i}} \sum_{0\le s<\ell+1} (x^{p^{e-(\ell+1)}})^{ap^s q^{in_\ell}} \\ &\quad -p \cdot \textstyle\sum_{0\le s<\ell} (x^{p^{e-\ell}})^{bp^s} \in \mathbb{F}_q[x]. \end{aligned} \tag{8.12}$$

Proposition 8.13. *For $0 < \ell < e$, let τ_0 and $\tau_{\ell,b}$ be the polynomials as in (8.12) for all $b \in \pi_{p^\ell}(\mathcal{K})$. Then Φ_{p^e} divides τ_0 and $\tau_{\ell,b}$.*

Proof. Fix $0 < \ell < e$, and let $\pi \colon \mathbb{Z}_{p^{\ell+1}}^\times \to \mathbb{Z}_{p^\ell}^\times$ with $\pi(a) = (a \bmod p^\ell)$ the canonical projection from $\mathbb{Z}_{p^{\ell+1}}^\times$ onto $\mathbb{Z}_{p^\ell}^\times$. Since we have $\pi_{p^\ell} = \pi \circ \pi_{p^{\ell+1}}$, the projection is a surjective homomorphism. Thus, each element $b \in \mathbb{Z}_{p^\ell}^\times$ has a preimage set $\pi^{-1}(b) = \{a \in \mathbb{Z}_{p^{\ell+1}}^\times \colon a \equiv b \bmod p^\ell\}$ of order $\#\pi^{-1}(b) = \frac{\#\mathbb{Z}_{p^{\ell+1}}^\times}{\#\mathbb{Z}_{p^\ell}^\times} = \frac{p^\ell(p-1)}{p^{\ell-1}(p-1)} = p$. One can easily check that the kernel of π is $\ker\pi = \{(1+p^\ell z) \bmod p^{\ell+1} \colon 0 \le z < p\}$. This gives a second way to express the preimage set of b in $\mathbb{Z}_{p^{\ell+1}}^\times$:

$$\pi^{-1}(b) = b \cdot \ker\pi = \{(b + zp^\ell) \bmod p^{\ell+1} \colon 0 \le z < p\}. \tag{8.14}$$

Here we use that the map $\psi_b \colon \{0,\dots,p-1\} \to \{0,\dots,p-1\}$ with $\psi_b(z) = bz \operatorname{rem} p$ is a permutation because $\gcd(b,p) = 1$.

We can also give a description of $\pi^{-1}(b)$ involving $\mathcal{I}_{\ell,b,i}$. Since we know that $q^{n_\ell} \in \pi_{p^\ell}(\mathcal{K})$, also the inverse of q^{in_ℓ} is an element in $\pi_{p^\ell}(\mathcal{K})$. Thus, the set $\mathcal{I}_{\ell,b,i}$ contains $\frac{k_{\ell+1}}{k_\ell}$ elements. For $0 < i < \frac{n_{\ell+1}}{n_\ell}$ and $a \in \mathcal{I}_{\ell,b,i}$, we have $\pi(q^{in_\ell} \cdot a) \equiv q^{in_\ell} \cdot q^{-in_\ell} b \equiv b \bmod p^\ell$. Hence, the set $\{q^{in_\ell}a \colon 0 \le i < \frac{n_{\ell+1}}{n_\ell} \text{ and } a \in \mathcal{I}_{\ell,b,i}\}$ is a subset of $\pi^{-1}(b)$. But $\biguplus_{0\le i<n_{\ell+1}} q^i \pi_{p^{\ell+1}}(\mathcal{K})$ is a partition of $\mathbb{Z}_{p^{\ell+1}}^\times$, and each subset has $\frac{n_{\ell+1}}{n_\ell} \cdot \frac{k_{\ell+1}}{k_\ell} = \frac{\phi(p^{\ell+1})}{\phi(p^\ell)} = p$ different elements. Therefore, equality holds:

$$\pi^{-1}(b) = \left\{ q^{in_\ell} a \colon 0 \le i < \frac{n_{\ell+1}}{n_\ell} \text{ and } a \in \mathcal{I}_{\ell,b,i} \right\}. \tag{8.15}$$

With the help of these formulas we have for $0 < \ell < e$ and all $b \in \pi_{p^\ell}(\mathcal{K})$:

$$\begin{aligned}
&\sum_{0\le i<\frac{n_{\ell+1}}{n_\ell}} \sum_{a\in\mathcal{I}_{\ell,b,i}} \sum_{0\le s<\ell+1} (x^{p^{e-(\ell+1)}})^{aq^{in_\ell}p^s} \\
&\overset{(8.15)}{\equiv} \sum_{a\in\pi^{-1}(b)} \sum_{0\le s<\ell+1} (x^{p^{e-(\ell+1)}})^{ap^s} \overset{(8.14)}{\equiv} \sum_{0\le z<p} \sum_{0\le s<\ell+1} (x^{p^{e-(\ell+1)}})^{p^s(b+zp^\ell)} \\
&\equiv \sum_{0\le s<\ell+1} \left((x^{p^{e-(\ell+1)}})^{bp^s} \cdot \left(\sum_{0\le z<p} (x^{p^{e-1}})^{zp^s} \right) \right) \bmod (x^{p^e}-1)
\end{aligned}$$

For $s = 0$, the sum in the inner brackets vanishes modulo Φ_{p^e} since

$$\sum_{0\le z<p} (x^{p^{e-1}})^z = \frac{x^{p^e}-1}{x^{p^{e-1}}-1} \equiv 0 \bmod \Phi_{p^e}$$

by (7.15). For $s \ge 1$, we simplify modulo Φ_{p^e}:

$$\sum_{0\le z<p} (x^{p^{e-1}})^{p^{1+(s-1)}} \equiv \sum_{0\le z<p} 1^{p^{s-1}} \equiv p \bmod \Phi_{p^e}.$$

Inserting both formulas gives

$$\begin{aligned}
&\sum_{0\le i<\frac{n_{\ell+1}}{n_\ell}} \sum_{a\in\mathcal{I}_{\ell,b,i}} \sum_{0\le s<\ell+1} (x^{p^{e-(\ell+1)}})^{ap^sq^{in_\ell}} \\
&\equiv (x^{p^{e-(\ell+1)}})^{bp^0} \cdot 0 + \sum_{1\le s<\ell+1} (x^{p^{e-(\ell+1)}})^{bp^s} \cdot p \\
&\equiv p \cdot \sum_{0\le s<\ell} (x^{p^{e-\ell}})^{bp^s} \bmod \Phi_{p^e}. \qquad (8.16)
\end{aligned}$$

It follows by construction of $\tau_{\ell,b}$ in (8.12) that Φ_{p^e} is a divisor of $\tau_{\ell,b}$ for $0 < \ell < e$ and $b \in \pi_{p^\ell}(\mathcal{K})$. For τ_0 we have

$$\begin{aligned}
&\sum_{0\le i<n_1} \sum_{a\in\pi_p(\mathcal{K})} (x^{p^{e-1}})^{aq^i} = \sum_{a\in\mathbb{Z}_p^\times} (x^{p^{e-1}})^a \\
&= \sum_{0\le z<p} (x^{p^{e-1}})^z - 1 = \frac{x^{p^e}-1}{x^{p^{e-1}}-1} - 1 \equiv -1 \bmod \Phi_{p^e}
\end{aligned}$$

since $\langle q, \pi_p(\mathcal{K})\rangle = \mathbb{Z}_p^\times$, and the claim follows also for τ_0 with (7.15). □

Let $\zeta^{p^{e-\ell}} = \zeta_\ell$ be a primitive p^ℓ-th root of unity for $0 \le \ell < e$. Since $e - \ell \ge 1$ and $(\zeta_\ell)^{p^{e-1}} = \zeta^{p^{e-\ell+e-1}} = 1$, a simple computation gives

$$\tau_0(\zeta_\ell) = \sum_{a\in\pi_p(\mathcal{K})} \sum_{0\le i<n_1} (\zeta_\ell^{p^{e-1}})^{aq^i} + 1 = n_1k_1 + 1 = \phi(p) + 1 = p \ne 0 \text{ in } \mathbb{F}_q.$$

Thus, $\gcd(\tau_0, \Phi_{p^\ell}) = 1$ for $0 \le \ell < e$ and for all $b \in \pi_{p^\ell}(\mathcal{K})$ we have

$$\gcd(\tau_0, \tau_{1,b}, \ldots, \tau_{e-1,b}, x^{p^e} - 1) = \Phi_{p^e} \text{ in } \mathbb{F}_q[x]. \tag{8.17}$$

Since $q^{in_\ell} \in \pi_{p^\ell}(\mathcal{K})$, we can write $\pi^{-1}(\pi_{p^\ell}(\mathcal{K}))$ as

$$\pi^{-1}(\pi_{p^\ell}(\mathcal{K})) = \pi_{p^{\ell+1}}(\mathcal{K}) = \biguplus_{0 \le i < \frac{n_{\ell+1}}{n_\ell}} \biguplus_{b \in \pi_{p^\ell}(\mathcal{K})} \mathcal{I}_{\ell,b,i}.$$

A direct consequence is that for $0 < \ell < e$

$$\begin{aligned} \tau_\ell = \; & \textstyle\sum_{0 \le i < \frac{n_{\ell+1}}{n_\ell}} \sum_{a \in \pi_{p^{\ell+1}}(\mathcal{K})} \sum_{0 \le s < \ell+1} (x^{p^{e-(\ell+1)}})^{ap^s q^{in_\ell}} \\ & -p \cdot \textstyle\sum_{b \in \pi_{p^\ell}(\mathcal{K})} \sum_{0 \le s < \ell} (x^{p^{e-\ell}})^{bp^s} \end{aligned} \tag{8.18}$$

is divisible by Φ_{p^e}.

REMARK 8.19. *Successively applying (8.12) and (8.18), respectively, we can transform the equation given in Lemma 8.5 into*

$$\left(\sum_{a \in \mathcal{K}} \sum_{0 \le s < e} x^{ap^s q^i} \right) \cdot \left(\sum_{b \in \mathcal{K}} \sum_{0 \le s' < e} x^{bp^{s'}} \right) \equiv \sum_{0 \le h < n} C'_{e,h} \cdot \left(\sum_{a \in \mathcal{K}} \sum_{0 \le s < e} x^{ap^s} \right)^{q^h} \bmod \Phi_{p^e}$$

where $C'_{e,h}$ depends on $0 \le i < n$.

This is indeed a way to compute a suitable preimage of $\varphi(A) \cdot \varphi(B)$ in $\mathcal{R} = \mathbb{F}_q[x]/(\Phi_{p^e})$. We observe that the final formula is due to a basis of $\mathcal{R}$ which supports the back-transformation into a linear combination of the conjugates of a normal Gauß period α.

LEMMA 8.20. *Let $\zeta = (x \bmod \Phi_{p^e})$ and $\mathcal{R} = \mathbb{F}_q[x]/(\Phi_{p^e})$. If $\mathbb{Z}_{p^e}^\times = \langle q, \mathcal{K} \rangle$ for a subgroup $\mathcal{K}$ of $\mathbb{Z}_{p^e}^\times$ then*

$$\mathcal{B} = \left\{ \sum_{0 \le s < e} \zeta^{ap^s} : a \in \mathbb{Z}_{p^e}^\times \right\}$$

is a basis of $\mathcal{R}$.

PROOF. A canonical basis of the polynomial ring $\mathcal{R}$ over $\mathbb{F}_q$ is the set $\mathcal{B}' = \{1, \zeta, \ldots, \zeta^{\phi(p^e)-1}\}$. Since $\mathcal{B}$ has at most $\#\mathcal{B}' = \phi(p^e)$ elements, it is sufficient to prove that $\mathcal{B}' \subseteq \langle \mathcal{B} \rangle$.

By construction, we have $\sum_{0 \le s < e} \zeta^{ap^s} \in \langle \mathcal{B} \rangle$ for $a \in \mathbb{Z}_{p^e}^\times$. By induction on ℓ, we find with (8.12) that for $0 < \ell < e$ we have $\sum_{0 \le s < \ell} (\zeta^{p^{e-\ell}})^{ap^s} \in \langle \mathcal{B} \rangle$ for $a \in \mathbb{Z}_{p^\ell}^\times$.

As a direct consequence of the previous and (8.12), we get $-1 \in \langle \mathcal{B} \rangle$.

Now let $1 \le a < \phi(p^e)$. Then there exist uniquely determined $0 < \ell \le e$ and $c \in \mathbb{Z}_{p^\ell}^\times$ such that $a = p^{e-\ell}c$, and

$$\sum_{0\le s<\ell} (\zeta^{p^{e-\ell}})^{cp^s} = \zeta^{cp^{e-\ell}} + \sum_{1\le s<\ell} (\zeta^{p^{e-\ell}})^{cp^s} = \zeta^{cp^{e-\ell}} + \sum_{0\le s<\ell-1} (\zeta^{p^{e-(\ell+1)}})^{cp^s}.$$

But both $\sum_{0\le s<\ell}(\zeta^{p^{e-\ell}})^{cp^s}$ and $\sum_{0\le s<\ell-1}(\zeta^{p^{e-\ell+1}})^{cp^s}$ are elements of $\langle\mathcal{B}\rangle$. Hence, $\zeta^{cp^{e-\ell}} = \zeta^a \in \langle\mathcal{B}\rangle$ for all $0 \le a < \phi(p^e)$ and the claim follows. □

Now we translate this result into the language of traces that has motivated the choice of $\tau_{\ell,b}$. Let $\mathrm{Tr}_{q^{n_\ell}/q^{n_{\ell-1}}}$ be the trace map of $\mathbb{F}_{q^{n_\ell}}$ into $\mathbb{F}_{q^{n_{\ell-1}}}$ for $0 < \ell \le e$; here $n_0 = 1$ by definition. The Galois group $\mathrm{Gal}(\mathbb{F}_{q^n}/\mathbb{F}_{q^{n_\ell}})$ is generated by the Frobenius automorphism σ, and we have

$$\mathrm{Tr}_{q^{n_\ell}/q^{n_{\ell-1}}}(\alpha_\ell) = \sum_{\sigma'\in\mathrm{Gal}(\mathbb{F}_{q^{n_\ell}}/\mathbb{F}_{q^{n_{\ell-1}}})} \sigma'(\alpha_\ell) = \sum_{0\le i<n_\ell/n_{\ell-1}} \alpha_\ell^{q^{in_{\ell-1}}}.$$

Since ζ is a root of Φ_{p^e}, we can apply (8.18) to $\alpha_\ell = \sum_{a\in\mathcal{K}_\ell}\sum_{0\le s<\ell}(\zeta^{p^{e-\ell}})^{ap^s}$. Then

$$\mathrm{Tr}_{q^{n_{\ell+1}}/q^{n_\ell}}(\alpha_{\ell+1}) = p\alpha_\ell \text{ for all } 1 \le \ell < e. \tag{8.21}$$

For τ_0, we simply have

$$\mathrm{Tr}_{q^{n_1}/q}(\alpha_1) = \sum_{0\le i<n_1}\sum_{a\in\mathcal{K}_1}(\zeta^{p^{e-1}})^{aq^i} = \sum_{a\in\mathbb{Z}_p^\times}\zeta^a = \frac{\zeta^{p^e}-1}{\zeta^{p^{e-1}}-1} - 1 = -1. \tag{8.22}$$

The trace map is transitive, i.e. $\mathrm{Tr}_{q^n/q^{n_\ell}}(\alpha) = \mathrm{Tr}_{q^{n_{\ell+1}}/q^{n_\ell}}(\mathrm{Tr}_{q^n/q^{n_{\ell+1}}}(\alpha))$, as stated in Fact 7.23. We use this to prove Lemma 8.11 by induction on $0 \le \ell \le e$. The case $\ell = 0$ is also called the *absolute trace*.

Proof (of Lemma 8.11). For $\ell = e$, we have $\mathrm{Tr}_{q^n/q^{n_e}}(\alpha) = \mathrm{Tr}_{q^{n_e}/q^{n_e}}(\alpha) = \alpha_e$ since $n = n_e$. Now we suppose that the claim is true for an $1 < \ell < e$. Then $\mathrm{Tr}_{q^n/q^{n_\ell}}(\alpha_{\ell+1}) = \mathrm{Tr}_{q^{n_{\ell+1}}/q^{n_\ell}}(p^{e-(\ell+1)}\alpha_{\ell+1}) = p^{e-(\ell+1)}\mathrm{Tr}_{q^{n_{\ell+1}}/q^{n_\ell}}(\alpha_{\ell+1}) \overset{(8.21)}{=} p^{e-(\ell+1)}(p\alpha_\ell)$. For $\ell = 0$ we get $\mathrm{Tr}_{q^n/q}(\alpha) = p^{e-1}\mathrm{Tr}_{q^{n_1}/q}(\alpha_1) \overset{(8.22)}{=} -p^{e-1}$ in the same way. □

We finally rewrite Remark 8.19 inserting the root ζ of Φ_{p^e}.

Remark 8.23. *The primitive p^e-th root of unity ζ is a zero of Φ_{p^e}, and we have*

$$\alpha^{q^i}\cdot\alpha = \sum_{0\le h<n} C'_{e,h}\alpha^{q^h}$$

for all $0 \le i < n$. The $C'_{e,h}$ depend on the given $0 \le i < n$. They are elements of the prime subfield $\mathbb{F}$ of $\mathbb{F}_q$ because $C^{(i)}_{p^{e-\ell}q^h} \in \mathbb{F}$ by Lemma 8.5 and all manipulations on the coefficients are done in $\mathbb{F}$. Thus, the multiplication matrix $T_\mathcal{N}$ has entries in $\mathbb{F}$.

The complete algorithm. Although we have presented all parts of the algorithm so far, we summarize the complete multiplication routine. We use an implementation-oriented language, since this emphasizes the operations in $\mathbb{F}_q$.

ALGORITHM 8.24. The prime power case.

Input: A normal prime power Gauß period α of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ with $\mathcal{K}$ a subgroup of $\mathbb{Z}_{p^e}^\times$ of order k, and two elements $A = \sum_{0 \le i < n} A_i \alpha^{q^i}$ and $B = \sum_{0 \le i < n} B_i \alpha^{q^i}$ of $\mathbb{F}_{q^n}$ with coefficients $A_i, B_i \in \mathbb{F}_q$, $0 \le i < n$.

Output: The product $C = \sum_{0 \le i < n} C_i \alpha^{q^i}$ of A and B with coefficients $C_i \in \mathbb{F}_q$, $0 \le i < n$.

Transformation from $\mathbb{F}_{q^n}$ into $\mathbb{F}_q[x]/(x^{p^e} - 1)$:

1. Set $A'_j = 0$ and $B'_j = 0$ for all $0 < j < p^e$.
2. For all $0 \le i < n$ and $a \in \mathcal{K}$ do set $j = aq^i \operatorname{rem} p^e$ and $A'_j = A_i$ and $B'_j = B_i$.
3. For $0 < \ell < e$ and all $i \in \mathbb{Z}_{p^{e-(\ell-1)}}^\times$ do
4. set $j = i \cdot p^\ell \operatorname{rem} p^e$ and compute $A'_j \leftarrow A'_j + A'_i$ and $B'_j \leftarrow B'_j + B'_i$.
5. Set $A' = \sum_{1 \le j < p^e} A'_j x^j$ and $B' = \sum_{1 \le j < p^e} B'_j x^j$.

Multiplication in $\mathbb{F}_q[x]/(x^{p^e} - 1)$:

6. Compute $C' = \sum_{2 \le j < 2p^e - 1} C'_j x^j \leftarrow A' \cdot B'$ with (fast) polynomial multiplication in $\mathbb{F}_q[x]$.
7. Reduce C' modulo $x^{p^e} - 1$: For $2 \le j < p^e - 1$ do compute $C'_j \leftarrow C'_j + C'_{j+p^e}$. Set $C'_0 = C'_{p^e}$ and $C'_1 = C'_{p^e+1}$. Set $C' = \sum_{0 \le j < p^e} C'_j x^j$.

Write the product as a sum of Gauß periods in $\mathbb{F}_q[x]/(x^{p^e} - 1)$:

8. Set $C_0 = C'_0$.
9. For all $0 < \ell \le e$ and $0 \le h < n$ do set $D^{(\ell)}_{\ell,h} = 0$ and $C_{e,h} = C'_{q^h}$.
10. For ℓ from $e - 1$ down to 1 do 11–14
11. For $0 \le h < n_\ell$ do
12. Compute $D^{(\ell)}_{\ell,h} \leftarrow \frac{k_{\ell+1}}{k_\ell} \cdot \sum_{0 \le j < \frac{n_{\ell+1}}{n_\ell}} (D^{(\ell+1)}_{\ell+1,h+jn_\ell} + C_{\ell+1,h+jn_\ell})$.
13. For $0 \le h < n_\ell$ do
14. compute $C_{\ell,h} \leftarrow C'_{p^{e-\ell}q^h} - D^{(\ell)}_{\ell,h}$.
15. Set $C'' = C_0 + \sum_{0 < \ell \le e} \sum_{0 \le h < n_\ell} C_{\ell,h} \left(\sum_{a \in \pi_{p^\ell}(\mathcal{K})} \sum_{0 \le s < \ell} (x^{p^{e-\ell}})^{ap^s} \right)^{q^h} \bmod (x^{p^e} - 1)$.

Reduction modulo $\Phi_{p^e} \in \mathbb{F}_q[x]$ applying the trace map:

16. For $0 \le h < n_1$ do compute $C_{1,h} \leftarrow C_{1,h} - C_0$.
17. For $1 \le \ell < e$ and $0 \le h < n_\ell$ do
18. For $0 \le i < \frac{n_{\ell+1}}{n_\ell}$ do compute $C_{\ell+1,h+in_\ell} \leftarrow C_{\ell+1,h+in_\ell} + p^{-1} \cdot C_{\ell,h}$.

Back-transformation from $\mathcal{R} = \mathbb{F}_q[x]/(\Phi_{p^e})$ into $\mathbb{F}_{q^n}$:

19. For $0 \le h < n$ do set $C_h = C_{e,h}$.
20. Return $C = \sum_{0 \le h < n} C_h \alpha^{q^h}$.

LEMMA 8.25. *Algorithm 8.24 works as specified.*

PROOF. The computation of the transformation in steps 1–5 is due to the definition of Gauß periods. The multiplication in steps 6–7 in $\mathbb{F}_q[x]/(x^{p^e}-1)$ generates a preimage of the product of $A \cdot B$. To compute the reduction modulo Φ_{p^e}, we apply the reordering of the summands according to Proposition 8.10 in steps 8–15. Notice that we compute only the $D_{\ell,h}^{(\ell)}$ for $1 \le \ell < e$ according to Remark 8.9(iii). These are sufficient to get all coefficients of Lemma 8.5, see (8.8). The reduction in steps 16–18 is done due to (8.12) and (8.18), respectively. Thus, we get the preimage of $A \cdot B$ in the ring $\mathcal{R} = \mathbb{F}_q[x]/(\Phi_{p^e})$ under the isomorphism χ as stated in Remark 8.19. The final back-transformation (steps 19–20) uses the fact that C is a linear combination of the conjugates of α as claimed in Remark 8.23. □

It remains to count the number of operations in $\mathbb{F}_q$. We recall that $n_\ell \le \phi(p^\ell)$ for $1 \le \ell \le e$. Furthermore, the telescope sum below is useful:

$$\sum_{1\le\ell\le e} \phi(p^\ell) = \sum_{1\le\ell\le e} (p^\ell - p^{\ell-1}) = p^e + \sum_{1\le\ell<e} p^\ell - \sum_{1\le\ell<e} p^\ell - p^0 = p^e - 1.$$

We have the following estimates for each part of the algorithm. We emphasize the prime case $e = 1$ since some steps are omitted in this special situation.

- The transformation (steps 1–5) is calculated with 2 additions for each $i \in \mathbb{Z}^\times_{p^{e-(\ell-1)}}$ where $0 < \ell < e$. This results in a total of at most

$$\sum_{0<\ell<e} 2\phi(p^{e-(\ell-1)}) = 2\sum_{2\le\ell\le e} \phi(p^\ell) = 2(p^e - 1 - \phi(p)) = 2p^e - 2p$$

  operations in $\mathbb{F}_q$. For the prime case $e = 1$ we have $2(p^e - p) = 0$ operations.

- Since both A' and B' have absolute coefficient zero, the multiplication modulo $x^{p^e} - 1$ in steps 6–7 can be done with

$$\mathsf{M}(p^e - 1) + (p^e - 3)$$

  operations. The second term is caused by additions. If $e = 1$ then $p - 1 = \phi(p) = nk$.

- The resorting of the summands in steps 8–15 is omitted for the prime case $e = 1$. Otherwise $e \ge 2$ and we may assume that $\frac{k_{\ell+1}}{k_\ell}$ is precomputed for

all $0 < \ell < e$. Then the number of operations is bounded by

$$\begin{aligned} & \sum_{1\le\ell<e} \sum_{0\le h<n_\ell} \left(1 + \frac{n_{\ell+1}}{n_\ell} - 1 + \sum_{0\le i<\frac{n_{\ell+1}}{n_\ell}} 1 + 1 \right) \\ = \; & 2 \sum_{1\le\ell<e} n_{\ell+1} + \sum_{1\le\ell<e} n_\ell \le 2 \sum_{2\le\ell\le e} \phi(p^\ell) + \sum_{1\le\ell<e} \phi(p^\ell) \\ = \; & 2(p^e - p) + p^{e-1} - 1. \end{aligned}$$

- The trace is applied in steps 16–18. Step 16 is executed for all $e \ge 1$ with $n_1 \le p - 1$ operations. For $e = 1$, we have $n_1 = n$. If $e \ge 2$ the subsequent iterative computation of the trace map in steps 17–18 can be done with

$$\sum_{1\le\ell<e} \sum_{0\le h<n_\ell} \sum_{0\le i<\frac{n_{\ell+1}}{n_\ell}} 2 = 2 \sum_{2\le\ell\le e} n_\ell \le 2 \sum_{2\le\ell\le e} \phi(p^\ell) = 2(p^e - p)$$

  further operations if we suppose p^{-1} to be precomputed.

- The back-transformation (steps 19–20) can be done without operations in $\mathbb{F}_q$.

We summarize this detailed cost analysis in the next theorem. We set $r = p^e$ and $\mathsf{M}(p^e) = O(p^e \log p^e \cdot \operatorname{loglog} p^e)$, i.e. we choose the fast multiplication algorithm for polynomials of Schönhage & Strassen (1971) and Schönhage (1977), respectively, as cited in Fact 5.8, and Result 8.1 follows immediately.

THEOREM 8.26. *Let q be a prime power co-prime to a prime p, and e a positive integer such that there exists a normal Gauß period α of type $(n, \mathcal{K})$ over $\mathbb{F}_q$, where $\mathcal{K}$ is a subgroup of $\mathbb{Z}_{p^e}^\times$. If the elements of $\mathbb{F}_{q^n}$ are given in the normal basis representation with respect to $\mathcal{N} = (\alpha, \ldots, \alpha^{q^{n-1}})$ then two elements can be multiplied with at most*

$$\mathsf{M}(p^e - 1) + 7p^e + p^{e-1} - 6p - 4 + n_1 \le \mathsf{M}(p^e) + 8p^e \in O(\mathsf{M}(p^e))$$

operations in $\mathbb{F}_q$.

We remark that all divisions that are computed in the algorithm (steps 12 and 18) are performed in the prime subfield $\mathbb{F}$ of $\mathbb{F}_{q^n}$. The only operations that are performed in $\mathbb{F}_q$ are additions/subtractions and multiplications.

The result of Gao *et al.* (2000) for the prime case is cited as a corollary. Blake *et al.* (1998) also described the idea to speed up multiplication in a normal basis representation using polynomial multiplication. The latter paper is restricted to optimal normal bases over $\mathbb{F}_2$ which are generated by prime Gauß periods with parameter $k \in \{1, 2\}$.

COROLLARY 8.27 (Gao *et al.* 2000, Theorem 4.1). *Let $\mathbb{F}_{q^n}$ be given by a normal basis $\mathcal{N} = (\alpha, \ldots, \alpha^{q^{n-1}})$, where α is a prime Gauß period of type (n, k) over $\mathbb{F}_q$. Then two elements of $\mathbb{F}_q$ given as a linear combination of the basis elements can be multiplied with at most*

$$\mathsf{M}(kn) + (k+1)n - 2$$

operations in $\mathbb{F}_q$.

Experiments. For prime and prime power Gauß periods, respectively, we ran two different test series. The test series *PrimeGP* contains prime Gauß periods of type (n, k) over $\mathbb{F}_2$ where $k \in \{1, 2\}$. The values for n are those of the other test series for polynomial and normal basis representation of $\mathbb{F}_{2^n}$ as described in Sections 5 and 6. Here, we want to verify that Gauß periods indeed offer a way to connect the advantages of both the polynomial and the normal basis representations as claimed in Result 8.1. In a second experiment, we compared prime and prime power Gauß periods. For prime power Gauß periods it is more expensive than for prime Gauß periods to get the suitable basis representation in $\mathcal{R} = \mathbb{F}_q[x]/(\Phi_{p^e})$; compare the estimates of Theorem 8.26 and Corollary 8.27. In particular, we discuss the rôle of the parameter k on this background.

All optimal normal bases of the test series *Normal* are generated by prime Gauß periods of type (n, k) with $k \in \{1, 2\}$. We copied these values for the test series *PrimeGP*. The routine for squaring in $\mathbb{F}_{2^n}$ was left unchanged, because all Gauß periods are normal. The matrix-based multiplication was replaced by Algorithm 8.24. The included polynomial multiplication is just the same as for polynomial basis representation: We used BIPOLAR's fast arithmetic library once more. In the case of the test series *PrimeGP* with $nk \leq 20000$, polynomial multiplication means mainly multiplication via Karatsuba with $\mathsf{M}(kn) = O((kn)^{\log_2 3})$. Thus, we have $\mathsf{M}(2n) = 3\mathsf{M}(n)$ if $k = 2$: a multiplication of a Gauß period of type $(n, 2)$ over $\mathbb{F}_2$ takes three times the time of a polynomial multiplication of two polynomials of degree less than n. We did 10000 trials to get the average times on multiplication and squaring for each field $\mathbb{F}_{2^n}$ of the test series *PrimeGP*. The times are documented in Figure 8.1, see also Table A.14. Obviously, the squaring is still free. Thus, changing the multiplication routine does not influence the possible speed-up on parallel exponentiation; $c = \frac{c_Q}{c_A}$ is still 0. Also the fast multiplication worked as predicted by the theory. For both $k = 1$ and $k = 2$ the costs were dominated by the polynomial multiplication. Figure 8.1 illustrates this by giving the relative time for one multiplication in terms of polynomial multiplications $\mathsf{M}(n)$; these blue dots are related to the right y-axis. The costs which are not caused by the polynomial multiplication (steps 6–7 of Algorithm 8.24) slow down the computation. But their proportion decreases for larger n. For $k = 1$, we have a total time that corresponds to 1.82 polynomial multiplications for $n = 1018$. For $n = 9802$ the corresponding value is only 1.16. For $k = 2$, the pairs are 3.90

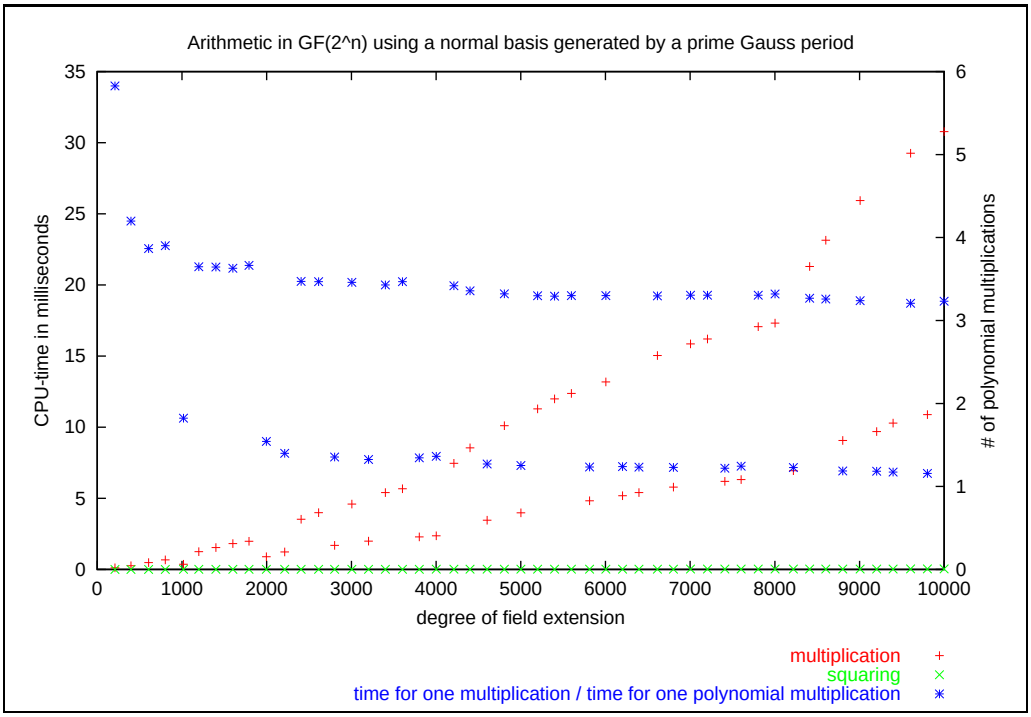


Figure 8.1: Times for multiplication (+) and squaring (×) for the test series *PrimeGP*. The times are the average of 10000 trials. Here $\mathbb{F}_{2^n}$ is given by a normal basis representation due to a normal basis $\mathcal{N} = (\alpha, \ldots, \alpha^{2^{n-1}})$, and α is a prime Gauß period of type (n, k) over $\mathbb{F}_2$ with $k \in \{1, 2\}$. The times are marked on the left y-axis. The scale of the right y-axis relates to the quotient (∗) between the time for one multiplication in this normal basis and the time for one polynomial multiplication of two polynomials of degree at most $n-1$ over $\mathbb{F}_2$.

for $n = 803$ and 3.23 for $n = 9998$. This coincides with the predicted quotient 3 for $k = 2$, for large n. The times for multiplication for $k = 1$ are comparable to those of the test series *Sparse*. Sparse polynomials offer the fastest arithmetic of the polynomial basis representations that have been presented in Section 5. The experiment illustrates that the parameter k is the crucial point for prime Gauß periods; this parameter is an indicator for the blow-up of the problem size which is caused by multiplying in the larger ring $\mathcal{R} = \mathbb{F}_q[x]/(x^p - 1)$ instead of $\mathbb{F}_{q^n}$.

Getting smaller values of k for given n and q was a motivation of Feisel *et al.* (1999) to generalize prime Gauß periods.[54] Comparing Table 6.1 and Table 8.2, we observe a slightly increase in the percentage of field extensions with small k. The prime power Gauß periods offer smaller values for k for exactly 105 values $2 \leq n < 10000$. The maximal decrease for the value of k is 30; we can substitute the prime Gauß period of type $(4374, 30)$ over $\mathbb{F}_2$ by the prime power Gauß period of

[54] "[...] the cost for arithmetic in $\mathbb{F}_{q^n}$ then depend not only on q and n but also on k. So it is important to find a value for k that is as small as possible.", Feisel *et al.* (1999), p. 1–2.

Existence of normal bases generated by a prime power Gauß period with given parameter $k \in \mathbb{N}_{\geq 1}$								
$k \setminus q$	2	3	5	7	11	13	17	19
$k = 1$	4.90	4.91	5.10	4.80	4.60	4.76	4.65	4.83
$k \leq 2$	17.58	18.86	18.15	17.78	17.64	17.71	17.30	17.65
$k \leq \log_2 n$	63.79	67.10	65.86	70.09	70.25	70.15	70.22	70.12
$k \leq \sqrt{n}$	86.96	91.12	89.40	95.84	97.08	95.45	96.29	97.85
$k < \infty$	87.51	91.67	90.01	96.43	97.73	96.16	97.06	98.69

Table 8.2: Percentage of field extensions $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ with $2 \leq n < 10000$ for which a normal basis generated by a prime power Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ exists. The parameter q is given in the top row; e.g. the figures for $\mathbb{F}_2$ are in the second column. The subgroup $\mathcal{K}$ of $\mathbb{Z}_{p^e}$ is of order $k \in \mathbb{N}_{\geq 1}$. The rows show the distribution if the value for k is also restricted. Note that we have limited our experiments for $r = p^e$ with $\phi(r) = p^{e-1}(p-1) = nk$ by $2 \leq r < 1000000$.

type $(4374, \{1\})$ where $\{1\} \subset \mathbb{Z}_{6561}^{\times}$. Both Gauß periods are normal in $\mathbb{F}_{2^{4374}}$ over $\mathbb{F}_2$. We collected some of the $n \in \{2, \ldots, 9999\}$ for which the prime power Gauß period of type $(n, \mathcal{K})$ is generated by a subgroup order $\#\mathcal{K} \leq 2$ and the parameter k for the prime Gauß period of type (n, k) is at least that order except for $n = 5050$. For this extension we compare a prime Gauß period of type $(5050, 1)$ over $\mathbb{F}_2$ with the prime power Gauß period of type $(5050, \{1 \bmod 10201, -1 \bmod 10201\})$. The times for this test series *PrimepowerGP* are shown in Figure 8.2. The exact values are documented in Table A.15. For both prime and prime power Gauß periods, multiplication is due to Algorithm 8.24. The times are the average over 10000 products of different factors $A, B \in \mathbb{F}_{2^n}$. The right y-axis marks the quotient of the order of the subgroup for the prime Gauß period versus the prime power Gauß period. The prime power Gauß period offers slower multiplication if the order of $\mathcal{K}$ is at least that of the prime Gauß period; this is the case for $n \in \{1014, 2028, 5050\}$. Though the r's for both prime and prime power Gauß periods are close to each other, the additional computation for prime power Gauß periods to identify the Gauß periods of the intermediate fields caused more computational effort. If the order of the subgroup is smaller in the prime power case, then the situation changes. The prime Gauß period is now clearly the inferior alternative. Prime power Gauß periods can do multiplication faster in this case—up to a factor of more than 14 for $n \in \{1830, 2211, 3660, 4422\}$. We conclude that k keeps its rôle; if a smaller k is available, then the substitution of a normal prime Gauß period by a prime power Gauß period for $\mathbb{F}_{2^n}$ speeds up field multiplication. Our experiments show that arithmetic benefits from our generalization of the approach of Gao *et al.* (2000)!

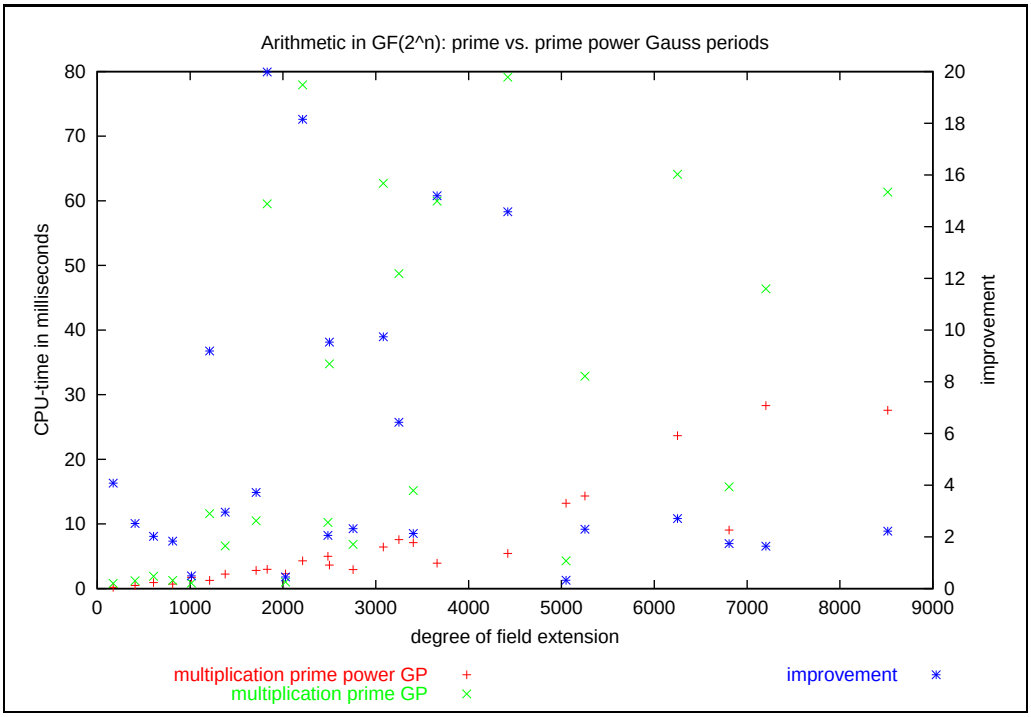


Figure 8.2: Times for multiplication for the test series *PrimepowerGP*. Prime power Gauß periods of type $(n, \mathcal{K})$ are marked by $+$, prime Gauß periods of type (n, k) by $\times$. The times are the average of 10000 trials and marked on the left y-axis. The scale of the right y-axis relates to the quotient $(*)$ between the order of the subgroup $\mathcal{K}$ for the prime power Gauß period and the order k of the subgroup for the prime Gauß period.

8.2. Algorithms for division. Since a normal prime power Gauß period α of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ determines a normal basis $\mathcal{N} = (\alpha, \alpha^q, \ldots, \alpha^{q^{n-1}})$ of $\mathbb{F}_{q^n}$, the algorithmic approach can be applied that has been described in Section 6.5 for *arbitrary* normal bases. This gives the bound for division in a normal basis representation due to $\mathcal{N}$ below. This follows directly from Theorem 6.24 combined with Theorem 8.26.

COROLLARY 8.28. *Let $\mathcal{N} = (\alpha, \ldots, \alpha^{q^{n-1}})$ be a normal basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ where α is a normal prime or prime power Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ with $\mathcal{K} \subseteq \mathbb{Z}_{p^e}^\times$. Let the elements be given by a normal basis representation with respect to $\mathcal{N}$. Then division by a non-zero element can be computed with*

$$O(\mathsf{M}(p^e) \cdot \log(nq)) \subseteq O(p^e \log p^e \cdot \operatorname{loglog} p^e \cdot (\log n + \log q))$$

operations in $\mathbb{F}_q$.

When we compute the inverse in the case of prime power Gauß periods, we must not stick to the application of *Fermat's Little Theorem*. We can also change to

the polynomial representation of the ring $\mathcal{R} = \mathbb{F}_{q^n}[x]/(\Phi_{p^e})$ using the homomorphism φ defined in (7.20). Then the inverse can be computed via the Extended Euclidean Algorithm 5.29. By working out this approach, we will prove the following consequence.

COROLLARY 8.29. *Let $\mathcal{N} = (\alpha, \ldots, \alpha^{q^{n-1}})$ be a normal basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$, where α a normal prime power Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ with $\mathcal{K} \subseteq \mathbb{Z}_{p^e}^\times$. Let the elements be given by a normal basis representation with respect to $\mathcal{N}$. Then the inverse of a non-zero element can be calculated with*

$$O(\mathsf{M}(p^e) \log p^e) \subseteq O(p^e \log^2 p^e \cdot \operatorname{loglog} p^e)$$

operations in $\mathbb{F}_q$.

To emphasize the rôle of the *Extended Euclidean Algorithm 5.29 (EEA)* for our algorithm, we define a function $\mathsf{E}\colon \mathbb{N}_{\geq 1} \to \mathbb{R}_{\geq 1}$: We set $\mathsf{E}(n)$ the number of operations in $\mathbb{F}_q$ to apply the EEA to two polynomials in $\mathbb{F}_q[x]$ of degree at most n.

Computing the inverse for prime Gauß periods. The idea to change to the polynomial representation is already described for optimal normal bases in $\mathbb{F}_{2^n}$, i.e. prime Gauß periods of type $(n, 1)$ and $(n, 2)$ over $\mathbb{F}_2$, by Rosing (1999), Section 11. He reports on a successful working implementation but omits a proof[55]. For all prime normal Gauß periods the same idea including a proof can be found in Gao *et al.* (2000).

FACT 8.30 (Gao *et al.* 2000, Theorem 4.1). *Let $\mathbb{F}_{q^n}$ be represented by a normal basis $\mathcal{N} = (\alpha, \ldots, \alpha^{q^n})$ where α is a prime Gauß period of type (n, k) over $\mathbb{F}_q$. Then division by a non-zero element in $\mathbb{F}_{q^n}$ can be computed with*

$$\mathsf{M}(nk) + (k+2)n - 2 + \mathsf{E}(nk) \in O(nk \log^2(nk) \operatorname{loglog}(nk))$$

operations in $\mathbb{F}_q$.

PROOF. Let $B = \sum_{0 \leq h < n} B_h \alpha^{q^h}$ be an element in $\mathbb{F}_{q^n}^\times$. We use the homomorphism $\varphi\colon \mathbb{F}_{q^n} \to \mathcal{R} = \mathbb{F}_q[x]/(\Phi_{p^e})$ with $\varphi(B) = \sum_{1 \leq i \leq nk} B'_i \zeta^i$ as defined in (7.20). Here $\zeta = (x \bmod \Phi_p)$ and $B'_i = B_h$ for $i \in q^h \mathcal{K}$. To compute B^{-1}, we note that $\varphi(B) \in \mathcal{R}$ is invertible modulo Φ_p. We can calculate the canonical representative of $\varphi(B)^{-1}$ in the basis $\mathcal{B}' = (1, \zeta, \ldots, \zeta^{nk-1})$ applying the Extended Euclidean Algorithm 5.29; this takes at most $\mathsf{E}(nk) \in O(\mathsf{M}(nk) \log(nk))$ operations in $\mathbb{F}_q$ due to Fact 5.30. To get B^{-1}, we have to represent $\varphi(B)^{-1}$ in the basis $\mathcal{B} = (\zeta, \ldots, \zeta^{nk})$ of $\mathcal{R}$. Changing between the bases $\mathcal{B} = (\zeta, \ldots, \zeta^{nk})$ and

[55] "I leave it to the mathematicians to prove it.", Rosing (1999), p. 286, and "I'm sure the mathematicians can prove this is true, because the code works!", Rosing (1999), p. 287.

$\mathcal{B}' = (1, \zeta, \ldots, \zeta^{nk-1})$ can be done with n additions in $\mathbb{F}_q$, using $\sum_{1 \le i < n} \zeta^i = -1$ as in Algorithm 8.24. Now we can substitute the division $A/B \in \mathcal{R}$ by the multiplication $A \cdot B^{-1}$. The computation of the product uses at most $\mathsf{M}(nk)+(k+1)n-2$ operations in $\mathbb{F}_q$ according to Corollary 8.27. □

EXAMPLE 8.31. Let ζ be a primitive 5-th root of unity. Then $\alpha = \zeta$ is the normal Gauß period of type $(n, \{1\})$ over $\mathbb{F}_2$ already discussed in Example 7.4(i). We have $\Phi_5 = \frac{x^5-1}{x-1} = x^4 + x^3 + x^2 + x + 1 \in \mathbb{F}_2[x]$. For the element $B = \alpha + \alpha^2 + \alpha^8$ in $\mathbb{F}_{2^4}$ we have $\varphi(B) = x + x^2 + x^3 \bmod \Phi_5$. This polynomial is invertible modulo Φ_5, and the Extended Euclidean Algorithm 5.29 computes $B' = x^3 + x + 1 \in \mathbb{F}_2[x]$ such that $B' \cdot B \equiv 1 \bmod \Phi_5$. We change from the canonical basis $\mathcal{B}' = (1 \bmod \Phi_5, x \bmod \Phi_5, x^2 \bmod \Phi_5, x^3 \bmod \Phi_5)$ to the basis $\mathcal{B} = (x \bmod \Phi_5, x^2 \bmod \Phi_5, x^4 \bmod \Phi_5, x^3 \bmod \Phi_5)$ by replacing $1 \equiv x + x^2 + x^4 + x^3 \bmod \Phi_5$. Thus, we get $B^{-1} = \varphi^{-1}(x^2 + x^4) = \alpha^2 + \alpha^4$. ◊

The prime power case. In the prime case we can easily compute the preferred basis representation of the inverse of B in $\mathcal{R} = \mathbb{F}_q[x]/(\Phi_p)$, since there is only the check whether the absolute coefficient of the representation of $\varphi(B^{-1})$ modulo $x^p - 1$ is non-zero. In the prime power case with $e \ge 2$, we first have to compute a representation of $\varphi(B^{-1}) \bmod x^{p^e-1}$ which can be transformed easily into a linear combination with respect to the basis $\mathcal{B} = \{\sum_{0 \le s < e} \zeta^{ap^s} : a \in \mathbb{Z}_{p^e}^\times\}$. The obvious idea is to do computations in the larger ring $\mathbb{F}_q[x]/(x^{p^e} - 1)$ instead of $\mathcal{R}$. This idea works well for multiplication. But it may fail for division because $\gcd(\varphi(B), x^{p^e} - 1)$ could be greater 1, i.e. $\varphi(B)$ might not be invertible in the larger ring.

EXAMPLE 8.32. Let ζ be a primitive 9-th root of unity. We recall the normal Gauß period $\alpha = \zeta + \zeta^3 + \zeta^8 + \zeta^6$ of type $(3, \{1, 8\})$ over $\mathbb{F}_2$ from Example 7.4(ii). If we apply the homomorphism φ to the element $B = \alpha + \alpha^2$ in $\mathbb{F}_{2^3}$ then $\varphi(B)$ is represented by the polynomial $x + x^8 + x^7 + x^2 \in \mathbb{F}_2[x]$. But $\gcd(x + x^8 + x^7 + x^2, x^9 - 1) = x^3 - 1 \in \mathbb{F}_2[x]$ is not trivial! ◊

By the Chinese Remainder Theorem, we have

$$\mathbb{F}_q[x]/(x^{p^e} - 1) \cong \mathbb{F}_q[x]/(x - 1) \times \mathbb{F}_q[x]/(\Phi_p) \times \cdots \times \mathbb{F}_q[x]/(\Phi_{p^e}),$$

since $x^{p^e} - 1$ factors over $\mathbb{F}_q$ into the pairwise co-prime polynomials $x-1$, Φ_p, …, Φ_{p^e} due to (7.16), and $x^{p^e-1} = (x-1) \cdot \prod_{1 \le \ell < e} \Phi_{p^\ell}$. Our algorithmic idea now is as follows: We first compute the inverse of $B \in \mathbb{F}_{q^n}$ in $\mathcal{R} = \mathbb{F}_q[x]/(\Phi_{p^e})$. The map $\varphi\colon \mathbb{F}_{q^n} \to \mathcal{R}$ as defined in (7.20) is an injective homomorphism. Hence, $\varphi(B)$ is invertible in $\mathcal{R}$ if and only if $B \in \mathbb{F}_{q^n}^\times = \mathbb{F}_q[x] \backslash \{0\}$. Then we compute the preimage of $(\varphi(B)^{-1}, 0) \in \mathbb{F}_q[x]/(\Phi_{p^e}) \times \mathbb{F}_q[x]/(x^{p^{e-1}} - 1)$ in the larger ring $\mathbb{F}_q[x]/(x^{p^e} - 1)$ using the Chinese Remainder Theorem. We claim that this preimage of B^{-1} in

$\mathbb{F}_q[x]/(x^{p^e}-1)$ can easily be transformed into a linear combination with respect to $\mathcal{B} = \left\{\sum_{0\le s<e} \zeta^{ap^s} : a \in \mathbb{Z}_{p^e}^\times\right\}$ by applying the trace map.

Now, we suppose that $\varphi(B)^{-1}$ is given with respect to $\mathcal{B}$, i.e. $\varphi(B^{-1}) = \sum_{0\le i<n} B_i \cdot \varphi(\alpha^{q^i})$ has the preimage $C = \sum_{0\le i<n} C_i \sum_{a\in\mathcal{K}} \sum_{0\le s<e} x^{ap^s q^i} \bmod (x^{p^e}-1)$ in the larger ring. Then $B^{-1} = \varphi^{-1}(C)$ can be computed without operations in $\mathbb{F}_q$. The image of C onto $\mathcal{R} = \mathbb{F}_q[x]/(\Phi_{p^e})$ is just $\varphi(B)^{-1}$. To compute the image of C onto $\mathbb{F}_q[x]/(x^{p^{e-1}}-1)$, it is sufficient to compute $\sum_{a\in\mathcal{K}} \sum_{0\le s<e} x^{ap^s} \bmod (x^{p^{e-1}}-1)$, since the canonical projection is a homomorphism. Then

$$\begin{aligned}\sum_{a\in\mathcal{K}} \sum_{0\le s<e} x^{ap^s} &= \sum_{a\in\mathcal{K}} \left(\sum_{0\le s<e-1} x^{ap^s} + x^{ap^{e-1}} \right) \\ &\equiv \frac{k}{k_{e-1}} \cdot \sum_{a\in\pi_{p^{e-1}}(\mathcal{K})} \sum_{0\le s<e-1} x^{ap^s} + k \cdot 1 \bmod (x^{p^{e-1}}-1).\end{aligned}$$

Thus, we have that $C \bmod x^{p^{e-1}-1}$ is given as a sum of Gauß periods in the subring $\mathbb{F}_q[x]/(x^{p^{e-1}}-1) \cong \mathbb{F}_q[x]/(x-1) \times \cdots \times \mathbb{F}_q[x]/(\Phi_{e-1})$. Our construction sets the projection onto $\mathbb{F}_q[x]/(x^{p^{e-1}}-1)$ to be 0 mod $(x^{p^{e-1}}-1)$. We successively apply τ_ℓ for $0 \le \ell < e$ as defined in (8.12) and (8.18), respectively, to compute $C \bmod (x^{p^{e-1}}-1)$. Since $\tau_\ell \equiv 0 \bmod \Phi_{p^e}$, the projection on $\mathcal{R}$ is not touched by this substitutions: Let $0 < \ell < e$. Then, we have

$$\begin{aligned}\tau_\ell &= \sum_{0\le i<\frac{n_{\ell+1}}{n_\ell}} \sum_{a\in\pi_{p^{\ell+1}}(\mathcal{K})} \sum_{0\le s<\ell+1} (x^{p^{e-(\ell+1)}})^{ap^s q^{in_\ell}} - p \cdot \sum_{b\in\pi_{p^\ell}(\mathcal{K})} \sum_{0\le s<\ell} (x^{p^{e-\ell}})^{bp^s} \\ &= \sum_{0\le i<\frac{n_{\ell+1}}{n_\ell}} \sum_{a\in\pi_{p^{\ell+1}}(\mathcal{K})} \left(\sum_{0\le s<\ell} (x^{p^{e-(\ell+1)}})^{ap^s q^{in_\ell}} + x^{p^{e-\ell-1}\cdot p^\ell a q^{in_\ell}} \right) \\ &\quad - p \cdot \sum_{b\in\pi_{p^\ell}(\mathcal{K})} \left(\sum_{0\le s<\ell-1} (x^{p^{e-\ell}})^{bp^s} + x^{p^{e-\ell}\cdot p^{\ell-1} b} \right) \\ &\equiv \sum_{0\le i<\frac{n_{\ell+1}}{n_\ell}} \frac{k_{\ell+1}}{k_\ell} \sum_{a\in\pi_{p^\ell}(\mathcal{K})} \sum_{0\le s<\ell} (x^{p^{e-(\ell+1)}})^{ap^s q^{in_\ell}} - p \cdot \sum_{b\in\pi_{p^\ell}(\mathcal{K})} \sum_{1\le s<\ell} (x^{p^{e-(\ell+1)}})^{bp^s} \\ &\quad + \frac{n_{\ell+1}}{n_\ell} \cdot k_{\ell+1} \cdot 1 - p \cdot k_\ell \bmod (x^{p^{e-1}}-1).\end{aligned}$$

But $q^{n_\ell} \in \pi_{p^\ell}(\mathcal{K})$ and $\frac{n_{\ell+1}k_{\ell+1}}{n_\ell k_\ell} = \frac{\phi(p^{\ell+1})}{\phi(p^\ell)} = p$, which gives

$$\begin{aligned}\tau_\ell &\equiv \frac{n_{\ell+1}k_{\ell+1}}{n_\ell k_\ell} \cdot \sum_{a\in\pi_{p^\ell}(\mathcal{K})} \sum_{0\le s<\ell} (x^{p^{e-(\ell+1)}})^{ap^s} - p \cdot \sum_{b\in\pi_{p^\ell}(\mathcal{K})} \sum_{1\le s<\ell} (x^{p^{e-(\ell+1)}})^{bp^s} \\ &\equiv p \cdot \sum_{a\in\pi_{p^\ell}(\mathcal{K})} (x^{p^{e-(\ell+1)}})^a \bmod (x^{p^{e-1}}-1).\end{aligned}$$

For $\ell = 0$, a similar result holds with

$$\begin{aligned}\tau_0 &= \sum_{0 \le i < n_1} \sum_{a \in \pi_p(\mathcal{K})} (x^{p^{e-1}})^{aq^{in_1}} + 1 \equiv \sum_{a \in \langle q, \mathcal{K} \rangle} (x^{p^{e-1}})^a + 1 \\ &\equiv \sum_{1 \le z < p} 1 + 1 = p \bmod (x^{p^{e-1}} - 1).\end{aligned}$$

By assumption, we have $\gcd(q, p) = 1$. Thus, we may apply $p^{-1} \cdot \tau_\ell$ for $0 \le \ell < e$. This shows that $C \bmod (x^{p^e} - 1)$ and $(\varphi(B)^{-1} \bmod \Phi_{p^e}, 0 \bmod x^{p^{e-1}} - 1)$ can be computed from each other by successively applying the trace map. This transformation has already been presented in detail in steps 8–18 of Algorithm 8.24. If $d' \in \mathbb{F}_q[x]$ of degree less than $\phi(p^e)$ is precomputed such that $D = d' \cdot (x^{p^{e-1}} - 1) \equiv 1 \bmod \Phi_{p^e}$ then the inverse of $B \in \mathbb{F}_{q^n}$ can be computed by using the Extended Euclidean Algorithm 5.29 once on Φ_{p^e} and B, computing the preimage $C = (\varphi(B)^{-1}, 0)$ with the Chinese Remainder Theorem using $d' \cdot (x^{p^{e-1}} - 1)$, and successively applying the trace map.

EXAMPLE 8.32 CONTINUED. We have $\Phi_9 = \frac{x^9-1}{x^3-1} = x^6 + x^3 + 1 \in \mathbb{F}_2[x]$. The Extended Euclidean Algorithm 5.29 computes on input Φ_9 and $\varphi(B) = x + x^2 + x^7 + x^8$ the output $C = x^5 + x^2 + x + 1 \in \mathbb{F}_2[x]$ such that $C \cdot (x + x^2 + x^7 + x^8) \equiv 1 \bmod \Phi_9$. Furthermore, we set $d = \frac{x^9-1}{\Phi_9} = x^3 - 1$; we can precompute $d' = x^3 \in \mathbb{F}_2[x]$ such that $D = d \cdot d' \equiv 1 \bmod \Phi_9$. This gives $C' = (C \cdot D) \equiv x^7 + x^6 + x^5 + x^4 + x^3 + x^2 \bmod x^9 - 1$. We sort this to see that C' is indeed a sum of Gauß periods: $C' = (x^2 + x^6 + x^7 + x^3) + (x^4 + x^3 + x^5 + x^6) + (x^3 + x^6) \in \mathbb{F}_2[x]$. Applying the trace map with $x^3 + x^6 \equiv x + x^8 + x^2 + x^7 + x^4 + x^5 + x^3 + x^6 \bmod \Phi_9$ results in $C' = x + x^8 + x^3 + x^6 \in \mathcal{R} = \mathbb{F}_q[x]/(\Phi_9)$, i.e. $B^{-1} = \alpha$. ◊

We give a formal algorithmic description to prove Corollary 8.29.

ALGORITHM 8.33. Inversion in the prime power case.

Input: A normal prime power Gauß period α of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ with $\mathcal{K}$ a subgroup of $\mathbb{Z}_{p^e}^\times$ of order k, and an element $B = \sum_{0 \le i < n} B_i \alpha^{q^i}$ with coefficients $B_i \in \mathbb{F}_q$ for $0 \le i < n$. A polynomial $D \in \mathbb{F}_q[x]$ of degree at most $p^e - 1$ with $D \equiv 0 \bmod x^{p^{e-1}} - 1$ and $D \equiv 1 \bmod \Phi_{p^e}$.

Output: The inverse B^{-1} of B in $\mathbb{F}_{q^n}$ as linear combination of the conjugates of α.

Transformation from $\mathbb{F}_{q^n}$ into $\mathbb{F}_q[x]/(x^{p^e} - 1)$:

1. Set $B'_j = 0$ for all $0 < j < p^e$.
2. For all $0 \le i < n$ and $a \in \mathcal{K}$ do set $j = aq^i \operatorname{rem} p^e$ and $B'_j = B_i$.
3. For $0 < \ell < e$ and all $i \in \mathbb{Z}^\times_{p^{e-(\ell-1)}}$ do
4. set $j = i \cdot p^\ell \operatorname{rem} p^e$ and compute $B'_j \leftarrow B'_j + B'_i$.
5. Set $B' = \sum_{1 \le j < p^e} B'_j x^j$.

Inversion in $\mathcal{R} = \mathbb{F}_q[x]/(\Phi_{p^e})$:

6. Compute $C = \sum_{0 \le j < \phi(p^e)} C_j x^j \in \mathbb{F}_q[x]$ such that $B' \cdot C = 1 \bmod \Phi_{p^e}$ by applying the Extended Euclidean Algorithm 5.29 on Φ_{p^e} and B'.
7. Compute $C' = \sum_{0 \le j < p^e + \phi(p^e) - 1} C'_j x^j \leftarrow D \cdot C$ with (fast) polynomial multiplication in $\mathbb{F}_q[x]$.
8. Reduce C' modulo $x^{p^e} - 1$: For $0 \le j < \phi(p^e) - 1$ do compute $C'_j \leftarrow C'_j + C'_{j+p^e}$. Set $C' = \sum_{0 \le j < p^e} C'_j x^j$.

Transform C' to get a suitable preimage of $\varphi(B)^{-1}$ in $\mathbb{F}_q[x]/(x^{p^e} - 1)$:

9. Apply Steps 8–18 of Algorithm 8.24 on C'. Let the result be C''.

Back-transformation from $\mathcal{R} = \mathbb{F}_q[x]/(\Phi_{p^e})$ into $\mathbb{F}_{q^n}$:

10. For $0 \le h < n$ do set $C_h = C''_{e,h}$.
11. Return $B^{-1} = \sum_{0 \le h < n} C_h \alpha^{q^h}$.

LEMMA 8.34. *Algorithm 8.33 computes the inverse of $B \in \mathbb{F}_{q^n}$ with at most $\mathsf{E}(p^e) + \mathsf{M}(p^e) + 6p^e - 4p - 2$ operations in $\mathbb{F}_q$.*

PROOF. Correctness follows with the arguments given above. For the counting of the number of operations for steps 1–5 and 9, we refer to the analysation of Algorithm 8.24. These steps can be done with $5p^e + p^{e-1} - 5p - 1 + n \le 5p^e + p^{e-1} - 4p - 2$ operations in $\mathbb{F}_q$. Step 6 can be performed with at most $\mathsf{E}(p^e)$ operations in $\mathbb{F}_q$ for $\deg B' < p^e$ and $\deg \Phi_{p^e} < \phi(p^e)$. In step 7, there is a multiplication of the polynomials C and D, both of degree less than p^e by construction. Thus, the product $C \cdot D$ can be computed with at most $\mathsf{M}(p^e)$ operations in $\mathbb{F}_q$. We count at most $\phi(p^e) = p^e - p^{e-1}$ operations in $\mathbb{F}_q$ in Step 8. We sum up the single estimates to get the claimed number of operations. □

PROOF (of Corollary 8.29). The claimed bound follows with Algorithm 8.33 if we apply fast polynomial arithmetic. We insert $\mathsf{E}(p^e) = O(\mathsf{M}(p^e) \log p^e)$ using Fact 5.32, and $\mathsf{M}(p^e) = O(p^e \log p^e \operatorname{loglog} p^e)$ due to Fact 5.8. This completes the proof. □

Experiments. In Section 6.5 we have reported on experiments on inversion in a normal basis representation. For the test series *Normal* the inversion algorithm is based on Fermat's Little Theorem 2.3. Multiplication for this test series is matrix-based multiplication. The experiments showed that the decrease on the number of steps in a star addition chain for $n - 1$ takes an significant effect on the total number of operations in $\mathbb{F}_{q^n}$.

We replicated these experiments for the test series *PrimeGP*. We computed the parallel inversion algorithm and compared it to the sequential inversion algorithm. The latter one ran on three different addition chains for $n - 1$ as input. The relation between the times of these four algorithms is still determined by the length of the star addition chains for $n - 1$. Figure 8.3 is very similar to Figure 6.4 if we ignore the absolute times. But in the case of prime Gauß periods

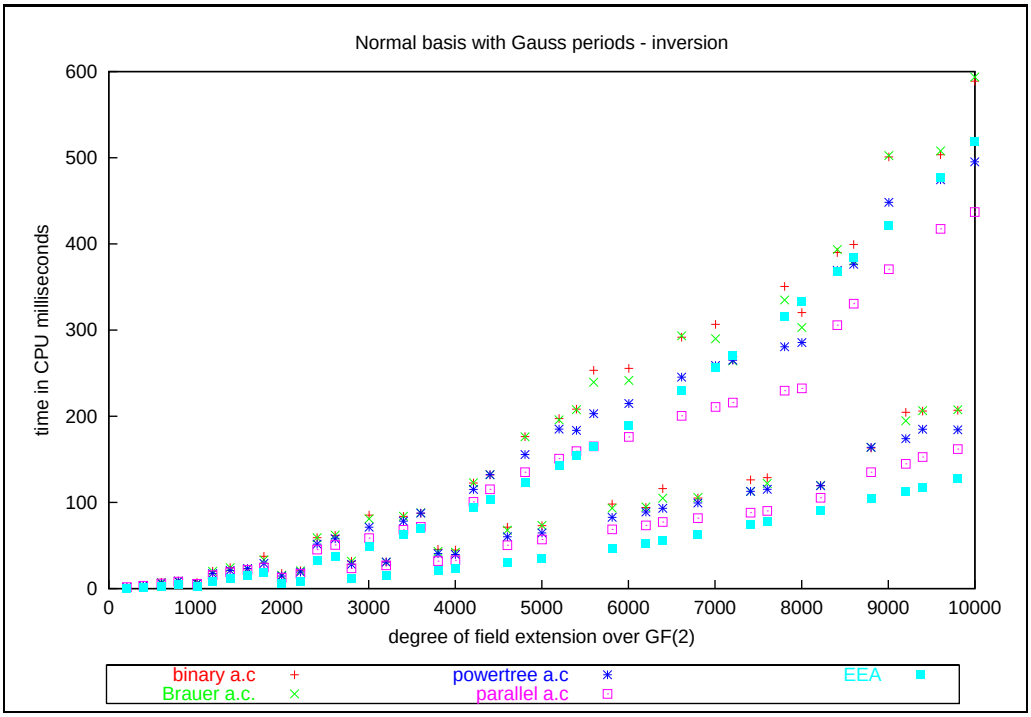


Figure 8.3: Times for inversion for the test series *PrimeGP*. The times are the average of 1000 trials. The graphical representation compares the sequential inversion algorithm a la Fermat (Algorithm 6.21) with the parallel version given in Algorithm 6.27 (□) and inversion using the Extended Euclidean Algorithm 5.29. The sequential version is performed on three different addition chains μ for $m = n - 1$ as input: the binary addition chain (+), Brauer addition chain (×), and the addition chain generated by the power tree algorithm (∗).

multiplication is computed according to Algorithm 8.24. Thus, a single multiplication is much faster than for the test series *Normal*. For $n = 209$, our new multiplication routine gave a speed-up of 16.27. This speed-up increased up to 303.07 for $n = 9802$ and 155.80 for $n = 9998$, see colum 8 in Table A.14. It depends on the parameter k. This improvement for a single multiplication implies nearly the same speed-up for inversion via Fermat. We compare sequential inversion when the binary addition chain μ for $n - 1$ is chosen. For $n = 209$, one inversion took 1.34 milliseconds using fast polynomial multiplication. The speed-up was roughly 15.05 compared to matrix-based multiplication, see column 5 of Table A.16. For $n = 9802$ and $n = 9998$, the improvements are 299.44 and 155.33, respectively. Since the number of multiplications for the test series *Normal* and *PrimeGP* are identical, the improvement by multiplication is the key to fast inversion via Fermat in a normal basis representation.

Additionally, we implemented inversion for prime power Gauß periods including the Extended Euclidean Algorithm 5.29. We simplified Algorithm 8.33 for

prime Gauß periods as described in the proof of Fact 8.30: we combined the polynomial multiplication in step 7 and the transformation in step 9 of Algorithm 8.33. We substituted these steps by n additions to change between the two different bases $\mathcal{B} = (\zeta, \ldots, \zeta^{nk})$ and $\mathcal{B}' = (1, \zeta, \ldots, \zeta^{nk-1})$. The times for inversion using the EEA are given in Figure 8.3 for the test series *PrimeGP*, see also Table A.17, column 6. We compared the times to those for inversion a la Fermat, see columns 7–10 of Table A.17. The EEA is faster for $k = 1$ and $n < 10000$ compared to parallel inversion (column 10). But the gain decreases from 2.03 for $n = 1018$ down to 1.18 for $n = 9802$. For $k = 2$ the break point is $n = 5399$; the parallel version beat the implementation of Algorithm 8.33 for all greater values of n. For all but one value of $n \geq 7005$ our experiments showed also better times for the fastest also given sequential variant of inversion a la Fermat, see Table A.17, column 9. Our experiments confirm the observation of Rosing (1999) for optimal normal bases in field extensions of small degree. But for larger $r = nk + 1$ our experiments showed that the approach using inversion due to Fermat's Little Theorem 2.3 is faster.

8.3. Computation of the multiplication matrix. For a normal prime Gauß period α of type (n, k) over $\mathbb{F}_q$, Wassermann (1990, 1993) gave an algorithm to compute the multiplication matrix $T_{\mathcal{N}}$ without applying polynomial arithmetic in $\mathbb{F}_q[x]$; he counted the elements in $(1 + q^i\mathcal{K}) \cap q^h\mathcal{K}$. These numbers—also called *cyclotomic numbers*[56]—are just our $u_{e,h}^{(i)}$ as defined in (8.3). Wassermann stressed the advantage to avoid the determination of an irreducible polynomial.[57] Beth *et al.* (1991) described a similar algorithm but used multiplication in $\mathbb{F}_q(\zeta)$. Our approach is a mixture of both ideas as suggested for the prime case by Gao *et al.* (2000).

By a careful rereading of the previous Section 8.1, we reveal an algorithm that takes up Wassermann's idea for normal prime power Gauß periods. The key to this algorithm is Proposition 8.7: for all $0 \leq i < n$, the coefficients of the product $\alpha^{q^i} \cdot \alpha$ only depend on the integers $u_{\ell,h}^{(i)}$ and $v_{\ell,h}^{(i)}$ as defined in (8.3). Our idea is now simple: determine all $u_{s,h}^{(i)}$ and $v_{s,h}^{(i)}$ using only arithmetic in $\mathbb{Z}_{p^e}$ and the prime subfield $\mathbb{F}$ of $\mathbb{F}_q$. Then we can calculate the coefficients as defined in Proposition 8.7. We may apply Algorithm 8.24 steps 8–20 on the sum

$$\alpha^{q^i} \cdot \alpha = C'_0 + \sum_{0<\ell\leq e} \sum_{0\leq h<n_\ell} C'_{p^{e-\ell}q^h} \sum_{a\in\pi_{p^\ell}(\mathcal{K})} (\zeta^{p^{e-\ell}})^{aq^h}$$

for all $0 \leq i < n$. Our algorithm will give the following result:

[56] See Gao *et al.* (2000), p. 884.

[57] "[...] insbesondere kann auf die Bestimmung eines irreduziblen Polynoms und auf die Überprüfung von linearer Unabhängigkeit verzichtet werden.", Wassermann (1993), p. 177.

RESULT 8.35. *Let α be a normal prime power Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ where $\mathcal{K}$ is a subgroup of $\mathbb{Z}_{p^e}^\times$. The multiplication matrix $T_\mathcal{N}$ of the normal basis $\mathcal{N} = (\alpha, \ldots, \alpha^{q^{n-1}})$ can be computed with at most*

$$2e\phi(p^e) + p^e + p^{e-1} + 2n + e - 6 \in O(ep^e)$$

operations in $\mathbb{Z}_{p^e}$ and at most

$$e\phi(p^e) + 4n\left(p^e + 2p^{e-1} + \frac{3}{4}e - p - \frac{11}{4}\right) \in O((e+n)p^e)$$

operations in the prime subfield $\mathbb{F}$ of $\mathbb{F}_q$.

Determination of $u_{\ell,h}^{(i)}$ and $v_{\ell,h}^{(i)}$. In Section 6.2 we have defined the multiplication matrix $T_\mathcal{N}$ which is a $n \times n$-matrix with entries in $\mathbb{F}_q$. In the case of normal prime power Gauß periods, the entries are indeed in the prime subfield $\mathbb{F}$ of $\mathbb{F}_q$ as stated in Remark 8.23. Therefore, we take an operation in this prime subfield as our basic operation. We additionally count the number of operations in $\mathbb{Z}_{p^e}$.

ALGORITHM 8.36. Determine the integers in (8.3).

Input: Integers $q, p, e, n \in \mathbb{N}_{\geq 1}$ with p a prime, q a prime power with $\gcd(q, p) = 1$, and a subgroup $\mathcal{K} \subseteq \mathbb{Z}_{p^e}^\times$ of order $k = \phi(p^e)/n$ such that $\langle q, \mathcal{K}\rangle = \mathbb{Z}_{p^e}^\times$.

Output: The set of integers $u_{\ell,h}^{(i)}$ and $v_{\ell,h}^{(i)}$ as defined in (8.3).

Precomputation: Determine $0 < \ell \leq e$ and $0 \leq h < n_\ell$ for all $b \in \mathbb{Z}_{p^e}^\times$ such that $b \in q^h \pi_{p^\ell}(\mathcal{K})$

1. For $0 < \ell \leq e$ do 2–4
2. If $\ell < e$ then compute $\pi_{p^\ell}(\mathcal{K}) = \{a \bmod p^\ell : a \in \pi_{p^{\ell+1}}(\mathcal{K})\}$.
3. For $0 \leq h < n_\ell$ do
4. For all $a \in \pi_{p^\ell}(\mathcal{K})$ do compute $b \leftarrow a \cdot (q^h \cdot p^{e-\ell}) \bmod p^e$ and store $\ell(b) = \ell$ and $h(b) = h$.
5. Store $\ell(0) = 0$ and $h(0) = 0$.

Compute the order of $(1 + q^i\mathcal{K}) \cap p^{e-\ell}q^h\mathcal{K}$ and $(1 + q^i p^\ell \mathcal{K}) \cap q^h\mathcal{K}$

6. For $0 \leq i < n$ do 7–10
7. Set $u_{\ell,h}^{(i)} = 0$ for all $0 \leq \ell \leq e$ and $0 \leq h < n_\ell$ and set $v_{\ell,h}^{(i)} = 0$ for all $1 \leq \ell < e$ and $0 \leq h < n_\ell$.
8. For all $a \in \mathcal{K}$ do 9–10
9. Compute $b \leftarrow (1 + aq^i) \bmod p^e$ and increment $u_{\ell(b),h(b)}^{(i)} \leftarrow u_{\ell(b),h(b)}^{(i)} + 1$.
10. For $0 < \ell < e$ compute $b \leftarrow (1 + ap^\ell q^i) \bmod p^e$ and increment $v_{\ell(b),h(b)}^{(i)} \leftarrow v_{\ell(b),h(b)}^{(i)} + 1$.
11. Return all $u_{\ell,h}^{(i)}$ and $v_{\ell,h}^{(i)}$.

PROPOSITION 8.37. *Algorithm 8.36 computes all* $u_{\ell,h}^{(i)}$ *and* $v_{\ell,h}^{(i)}$ *as given in (8.3) with at most* $2e\phi(p^e)+p^e+\sum_{0<\ell\le e} n_\ell+n+e-5 \le p^{e-1}(p+1)+2e\phi(p^e)+2n+e-6$ *operations in* $\mathbb{Z}_{p^e}$ *and* $e\phi(p^e)$ *operations in the prime subfield* $\mathbb{F}$ *of* $\mathbb{F}_q$.

PROOF. To generate the table in steps 1–5, we first compute $q^2,\ldots,q^{n-1}$ and $p^2,\ldots,p^{e-1}$ modulo p^e with $n-2+e-2$ multiplications in $\mathbb{Z}_{p^e}$. The precomputation of $q^h \cdot p^{e-\ell} \bmod p^e$ for all $0<\ell\le e$ and $0\le h<n_\ell$ takes $\sum_{0<\ell\le e}\sum_{0\le h<n_\ell} 1 = \sum_{0<\ell\le e} n_\ell \le n+\sum_{0<\ell<e}\phi(p^\ell) = n+p^{e-1}-1$ further operations in $\mathbb{Z}_{p^e}$. The three loops in steps 1–4 perform $\sum_{0<\ell\le e}\sum_{0\le h<n_\ell}\sum_{a\in\pi_{p^\ell}(\mathcal{K})} 1 = \sum_{0<\ell\le e} n_\ell k_\ell = \sum_{0<\ell\le e}\phi(p^\ell) = p^e-1$ additional multiplications. The computation of $\pi_{p^\ell}(\mathcal{K})$ for $1\le\ell\le e$ takes $\sum_{0<\ell<e} k_{\ell+1} \le \sum_{0<\ell\le e}\phi(p^\ell) = p^e-1$ reductions which we do not count. For the second part of the algorithm (steps 6–11) we distinguish between operations in $\mathbb{Z}_{p^e}$ and operations in the prime subfield $\mathbb{F}$ of $\mathbb{F}_q$. For the manipulations modulo p^e, we suppose that $p^\ell q^i \in \mathbb{Z}_{p^e}$ is already precomputed. Then, we count $\sum_{0\le i<n}\sum_{a\in\mathcal{K}}(2+\sum_{0<\ell<e} 2) = 2nk+2(e-1)nk = 2e\phi(p^e)$ operations in $\mathbb{Z}_{p^e}$. Similar, we have less than $nk+(e-1)nk = e\cdot\phi(p^e)$ operations in $\mathbb{F}$ to increment $u_{\ell,h}^{(i)}$ and $v_{\ell,h}^{(i)}$. □

Determination of the coefficients of $\alpha^{q^i}\cdot\alpha$. If the $u_{\ell,h}^{(i)}$ and $v_{\ell,h}^{(i)}$ are given then it is easy to determine the coefficients in Proposition 8.7. To avoid indexing with i, the next part of the algorithm takes input $0\le i<n$.

ALGORITHM 8.38. Determine the coefficients of Proposition 8.7.

Input: An integer $0\le i<n$ and $u_{\ell,h}^{(i)}$ and $v_{\ell,h}^{(i)}$ as determined by (8.3).
Output: The coefficients C'_0 and $C'_{p^{e-\ell}q^h}$ as determined by Proposition 8.7.

1. Set $C'_a = 0$ for all $0\le a<p^e$. Set $U_h = u_{e,h}^{(i)}$ and $V_h = 0$ and $C'_{q^h} = U_h$ for all $0\le h<n$.
2. For $0<\ell<e$ do
3. For $0\le h<n_{e-\ell}$ do compute $U_h \leftarrow U_h + u_{e-\ell,h}^{(i)}$ and $C'_{p^\ell q^h} \leftarrow C'_{p^\ell q^h} + U_h$.
4. For $0<\ell<e$ do
5. For $0\le h<n_\ell$ do compute $V_h \leftarrow V_h + (v_{\ell,h}^{(i)} + v_{\ell,h-i}^{(n-i)})$ and $C'_{p^{e-\ell}q^h} \leftarrow C'_{p^{e-\ell}q^h} + V_h$.
6. For $0<\ell<e$ and $0\le h<n_\ell$ compute $C'_{p^{e-\ell}q^h} \leftarrow \frac{k}{k_\ell}\cdot C'_{p^{e-\ell}q^h}$.
7. If $u_{0,0}^{(i)} = 1$ then set $C'_0 = e$.
8. For $0<\ell<e$ do 9–11
9. Set $C = u_{\ell,0}^{(i)}$.
10. For $0<h<n_\ell$ do compute $C \leftarrow C + u_{\ell,h}^{(i)}$.
11. Compute $C'_0 \leftarrow C'_0 + C\cdot(e-\ell)$.
12. Compute $C'_0 \leftarrow k\cdot C'_0$.
13. Return C'_a for $0\le a<p^e$.

We may label the coefficient $C'_{p^{e-\ell}q^h}$ with the two indices ℓ and h. Then we can refer to all coefficients without index arithmetic. Therefore, the algorithm has only operations in the prime subfield $\mathbb{F}$.

PROPOSITION 8.39. *Let $0 \le i < n$ be an integer. Algorithm 8.38 computes the coefficients of $\alpha^{q^i} \cdot \alpha$ as given by Proposition 8.7 with at most $7 \cdot \sum_{0<\ell<e} n_\ell + 3e - 3 \le 7p^{e-1} + 3e - 10$ operations in the prime field $\mathbb{F}$ of $\mathbb{F}_q$*

PROOF. Correctness follows with the formulas of Proposition 8.7. We count the number of operations: The first loop (steps 2–3) uses $\sum_{0<\ell<e} \sum_{0<h<n_{e-\ell}} 2 = 2\sum_{0<\ell<e} n_\ell \le 2\sum_{0<\ell<e} \phi(p^\ell) = 2(p^{e-1} - 1)$ operations. Summing up V_h in steps 4–5 can be done with at most $\sum_{0<\ell<e} \sum_{0\le h<n_\ell} 3 = 3\sum_{0<\ell<e} n_\ell$ further operations, which is bounded by $3\sum_{0<\ell<e} \phi(p^\ell) \le 3(p^{e-1} - 1)$. If we precompute $\frac{k}{k_\ell}$ for $0 < \ell < e$ in step 6 with $e - 1$ divisions then the step itself causes another $\sum_{0<\ell<e} n_\ell \le p^{e-1} - 1$ operations. Note that $k_e = k$ and therefore the case $\ell = e$ is omitted in this loop. The final computation of C'_0 causes $\sum_{0<\ell<e} n_\ell + 2(e-1) + 1 \le 2e - 2 + p^{e-1}$ operations in $\mathbb{F}$. □

Now, we can apply steps 8–20 of Algorithm 8.24 to compute the coefficients C_h of the product $\alpha^{q^i} \cdot \alpha = \sum_{0\le h<n} C_h \alpha^{q^h}$ for all $0 \le i < n$. The cost analysis of Algorithm 8.24 includes a bound of at most $4p^e + p^{e-1} - 4p - 1$ operations for each i. Adding the estimates of the previous two propositions proves Result 8.35. For the prime case $e = 1$ we get the following estimate. A similar result has been given by Wassermann (1993), Section 3.2.[58]

COROLLARY 8.40. *Let α be a normal Gauß period of type (n, k) over $\mathbb{F}_q$, and $\mathcal{K}$ be the unique subgroup of $\mathbb{Z}^\times_{nk+1}$ of order k. The multiplication matrix $T_\mathcal{N}$ of the normal basis $\mathcal{N} = (\alpha, \ldots, \alpha^{q^{n-1}})$ can be computed with at most $(3k + 2)n - 3$ operations in $\mathbb{Z}_{nk+1}$ plus nk operations in the prime subfield $\mathbb{F}$ of $\mathbb{F}_q$.*

8.4. Normality of prime power Gauß periods. We have already cited the Normal Gauß period theorem 7.5: a general Gauß period α of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ with $\mathcal{K} \subseteq \mathbb{Z}^\times_r$ is normal in $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ if and only if $\langle q, \mathcal{K}\rangle = \mathbb{Z}^\times_r$.

In this paragraph, we give a new proof for the case $r = p^e$. Thus, let α be a prime power Gauß period. We use the trace argument once more. Our tools are restricted to extensions over $\mathbb{F}_q$ of finite degree.

THEOREM 8.41. *Let p be a prime, e be a positive integer, and let α be a prime power Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ with $\mathcal{K}$ a subgroup of $\mathbb{Z}^\times_{p^e}$. Then α is normal in $\mathbb{F}_{q^n}$ if and only if $\langle q, \mathcal{K}\rangle = \mathbb{Z}^\times_{p^e}$.*

[58] "Für die Berechnung der $l_{v,j}$ wird ein Aufwand von $O(kn)$ benötigt. Zusammen mit der Reduktion von q modulo $kn + 1$ benötigt die Berechnung der Matrix T, abgesehen von der Vorbesetzung mit 0, also $O(kn + \ln q)$ Operationen.", Wassermann (1993), p. 185. [the $l_{v,j}$ are our $u^{(i)}_{0,h}$, and T is $T_\mathcal{N}$ in our notation.]

PROOF. By Remark 7.7, the condition $\langle q, \mathcal{K}\rangle = \mathbb{Z}_{p^e}^{\times}$ is necessary for a prime power Gauß period α to be normal in $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$. It remains to show that this condition is also sufficient. Since $\alpha^{q^n} = \alpha$, we have $\langle \alpha, \ldots, \alpha^{q^{n-1}}\rangle$ a subspace of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$. We prove that α and its conjugates are linearly independent. Since we have $\dim_{\mathbb{F}_q}(\mathbb{F}_{q^n}) = n$, the claim follows immediately.

We recall that φ is $\mathbb{F}_q$-linear. Thus, we have $\sum_{0\le h<n} u_h \alpha^{q^h} = 0$ if and only if $\sum_{0\le h<n} u_h \sum_{a\in\mathcal{K}} \sum_{0\le s<e} \zeta^{ap^s q^h} = 0$ in $\mathcal{R}$ with $\zeta = (x \bmod \Phi_{p^e})$. Since the subsets $q^h\mathcal{K}$ for $0 \le h < n$ form a partition of $\mathbb{Z}_{p^e}^{\times}$, we have

$$0 = \sum_{0\le h<n} u_h \sum_{a\in q^h\mathcal{K}} \left(\sum_{0\le s<e} \zeta^{ap^s}\right) = \sum_{a\in\mathbb{Z}_{p^e}^{\times}} u'_a \left(\sum_{0\le s<e} \zeta^{ap^s}\right)$$

and $\{u_0, \ldots, u_{n-1}\} \subseteq \{u'_a : a \in \mathbb{Z}_{p^e}^{\times}\}$. But $\mathcal{B} = \{\sum_{0\le s<e} \zeta^{ap^s} : a \in \mathbb{Z}_{p^e}^{\times}\}$ is a basis of $\mathcal{R}$ as proven in Lemma 8.20. This gives $u'_a = 0$ for all $a \in \mathbb{Z}_{p^e}^{\times}$. We conclude that $u_0 = \cdots = u_{n-1} = 0$. □

8.5. Exponentiation using normal prime power Gauß periods. Both the theoretical estimates and the experiments confirm that normal prime or prime power Gauß periods are the data structure we are looking for. This data structure connects the advantages of normal and polynomial basis representations if the parameter k is small. The disadvantage is that this preferred data structure does not exist for all finite fields $\mathbb{F}_{q^n}$, as illustrated in Table 8.2. Let us assume

Normal basis representation with a normal prime power Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_2$			
	classical	Karatsuba	FFT
arithmetic			
mult. (c_A)	$O(k^2n^2)$	$O(k^{1.59}n^{1.59})$	$O(kn\log n\,\mathrm{loglog}\,n)$
squaring (c_Q)	0	0	0
$c = \frac{c_Q}{c_A}$	0	0	0
exponentiation			
sequential	$O(k^2\frac{n^3}{\log_2 n})$	$O(k^{1.59}\frac{n^{2.59}}{\log n})$	$O(kn^2\,\mathrm{loglog}\,n)$
parallel	$O((kn)^2\log n)$	$O((kn)^{1.59}\log n)$	$O(kn\log^2 n\,\mathrm{loglog}\,n)$

Table 8.5: Bounds for exponentiation in $\mathbb{F}_{2^n}$ given by a normal basis representation generated by a prime power Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_2$ with $\mathcal{K}$ a subgroup of $\mathbb{Z}_r^{\times}$ of order k. We assume that $r \approx \phi(r) = nk$ and that k is polynomial bounded in n.

that $r \approx \phi(r)$, and let us assume that there is a prime or prime power Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_2$ with $\mathcal{K} \subseteq \mathbb{Z}_r^{\times}$ of order $k \in O(\log n)$. Then fast

arithmetic and the quotient $\frac{c_Q}{c_A} = 0$ result in soft-linear asymptotic upper bounds for parallel exponentiation in $\mathbb{F}_{2^n}$, see Table 8.5, column "FFT". This assumption holds for roughly 64% of all field extensions over $\mathbb{F}_2$ of maximal degree $n < 10000$. The three columns of Table 8.5 list the bounds for arithmetic using three different polynomial multiplication algorithms: classical multiplication with $\mathsf{M}(n) \in O(n^2)$ (Remark 5.2), Karatsuba's algorithm with $\mathsf{M}(n) \in O(n^{\log_2 3})$ (Corollary 5.7), and FFT-based multiplication with $\mathsf{M}(n) = O(n \log n \operatorname{loglog} n)$ (Fact 5.8). If $\mathsf{M}(n)$ is sub-quadratic then the bound on parallel exponentiation is also sub-quadratic! This beats all bounds for polynomial basis representations discussed in Section 5. All these data structures have bounds which are at least quadratic in n. Thus, the asymptotic estimates for Gauß periods are superior if k can be chosen small. This has already been shown for prime Gauß periods by Gao *et al.* (2000). Our results hitherto extend this result to prime power Gauß periods! We conclude that a normal basis for $\mathbb{F}_{q^n}$ generated by a suitable prime or prime power Gauß period meets exactly the requirements on a data structure that supports parallel exponentiation!

9. Decomposable Gauß periods

We now extend the result of Section 8 to the case that the Gauß period α of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ is neither a prime nor a prime power Gauß period. Our approach is to reduce the general case to the prime power case along the prime power decomposition of r. In this section we prove the following result.

RESULT 9.1. *Let r be a positive integer greater than 1 with prime power decomposition $r_1 \cdots r_t$, and let α' be a normal Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ with $\mathcal{K}$ a subgroup of $\mathbb{Z}_r^\times$. Then there is a Gauß period α such that $\mathcal{N} = (\alpha, \ldots, \alpha^{q^{n-1}})$ is a normal basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$. Two elements given by a linear combination of the elements of $\mathcal{N}$ can be multiplied with at most $O(\prod_{1 \le i \le t}(r_i \log r_i \cdot \operatorname{loglog} r_i))$ operations in $\mathbb{F}_q$.*

We derive this result in three steps. First, we introduce our tool: *decomposable Gauß periods*. Those Gauß periods can be written as a product of prime and prime power Gauß periods, respectively, see Section 9.1. We apply Algorithm 8.24 to this special type of Gauß periods as a second step in Section 9.2: the factorization of decomposable Gauß periods points a way to multiply along a tower of fields. Each multiplication in this tower appears to be the prime power case discussed in the previous section. As a third step, it remains to prove that normal decomposable Gauß periods exist whenever normal general Gauß periods do. This follows with a result of Gao (2001). We give the proof in Section 9.3. The closing Section 9.4 describes a criterion for the existence of normal general Gauß periods, which is mainly a generalization of a result of Gao *et al.* (2000). A similar approach can also be found in the paper of Gao (2001).

9.1. Decomposition of Gauß periods. Let α be a general Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ such that $\langle q, \mathcal{K} \rangle = \mathbb{Z}_r^\times$. By the Chinese Remainder Theorem, we can write $\mathbb{Z}_r^\times$ as a direct sum[59] along the prime power decomposition of $r = r_1 \cdots r_t$:

$$\mathbb{Z}_r^\times \cong \mathbb{Z}_{r_1}^\times \times \cdots \times \mathbb{Z}_{r_t}^\times.$$

Applying the canonical projections $\pi_{r_i} \colon \mathbb{Z}_r^\times \to \mathbb{Z}_{r_i}^\times$ for all $1 \le i \le t$ on the subgroup $\mathcal{K}$ gives

$$\mathcal{K} \subseteq \pi_{r_1}(\mathcal{K}) \times \cdots \times \pi_{r_t}(\mathcal{K}). \tag{9.2}$$

We have $\mathcal{K}$ a subgroup of the direct sum of its projections onto $\mathbb{Z}_{r_i}^\times$, $1 \le i \le t$. Unfortunately, equality does not always hold!

[59] Subsequently, we will identify both isomorphic groups and often write $=$ instead of $\cong$.

EXAMPLE 9.3. Recall the two subgroups $\mathcal{K}_1 = \{1, 26\}$ and $\mathcal{K}_2 = \{1, 44\}$ of $\mathbb{Z}_{45}^\times$ of order 2 in Example 7.4(iii). Both generate normal Gauß periods in $\mathbb{F}_{2^{12}}$ over $\mathbb{F}_2$. The first subgroup has the images $\pi_9(\mathcal{K}_1) = \{1, 8\}$ in $\mathbb{Z}_9^\times$ and $\pi_5(\mathcal{K}_1) = \{1\}$ in $\mathbb{Z}_5^\times$. For $\mathcal{K}_2$ the images are $\pi_9(\mathcal{K}_2) = \{1, 8\}$ and $\pi_5(\mathcal{K}_2) = \{1, 4\}$. Thus, $\mathcal{K}_1$ is the direct sum of its projected images while $\pi_9(\mathcal{K}_2) \times \pi_5(\mathcal{K}_2) \cong \{1, 19, 26, 44\} \neq \mathcal{K}_2$. $\Diamond$

If equality holds in (9.2) then we can write the Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ as a product of prime and prime power Gauß periods along the product $r_1 \cdots r_t$.

DEFINITION 9.4. *Let r be a positive integer greater than 1 with prime power decomposition $r_1 \cdots r_t$, and let $\mathcal{K}$ be a subgroup of $\mathbb{Z}_r^\times$.*

(i) Let $\pi_{r_i} \colon \mathbb{Z}_r^\times \to \mathbb{Z}_{r_i}^\times$ for $1 \leq i \leq t$ be the canonical projection. The subgroup $\mathcal{K}$ is called decomposable *if*

$$\mathcal{K} = \pi_{r_1}(\mathcal{K}) \times \cdots \times \pi_{r_t}(\mathcal{K}).$$

(ii) A Gauß period α of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ is decomposable *if $\mathcal{K}$ is decomposable.*

Let R_1 be the square-free part of r as in Definition 7.2[60]. We call a Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ with $\mathcal{K} \subseteq \mathbb{Z}_r^\times$ *square-free* if r is square-free, i.e. $r = R_1$. If $R_1 = 1$ then we call α *non-squarefree*.

A product of prime (power) Gauß periods. If equality is given in (9.2) then the factorization of r causes a factorization of a normal Gauß period α.

LEMMA 9.5. *Let α be a decomposable normal Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ with respect to ζ. Let $r_1 \cdots r_t$ be the prime power decomposition of r. For $1 \leq i \leq t$ let α_i be the Gauß period of type $(n_i, \pi_{r_i}(\mathcal{K}))$ over $\mathbb{F}_q$ with respect to $\zeta_i = \zeta^{r/r_i}$, where $n_i = \frac{\phi(r_i)}{\#\pi_{r_i}(\mathcal{K})}$. Then there are $0 \leq h_i < n_i$ for $1 \leq i \leq t$ such that*

$$\alpha = \prod_{1 \leq i \leq t} \alpha_i^{q^{h_i}}.$$

Before we give the prove, we illustrate this by an example.

[60] This definition of *square-free* may be different from the definition the reader is familiar with. Due to the definition here, the square-free part of $45 = 3^2 \cdot 5$ is 5, and not $3 \cdot 5$.

EXAMPLE 9.3 CONTINUED. Let ζ be a primitive 45-th root of unity. The normal Gauß period $\alpha = \zeta^{14} + \zeta^{24} + \zeta^4 + \zeta^{39}$ of type $(12, \{1, 26\})$ with $\{1, 26\} \subset \mathbb{Z}_{45}^\times$ is decomposable. The canonical projections along the prime power decomposition of $45 = 3^2 \cdot 5$ generate the prime Gauß period $\alpha_5 = \zeta^9$ of type $(4, \{1\})$ and the prime power Gauß period $\alpha_9 = \zeta^5 + (\zeta^5)^3 + (\zeta^5)^8 + (\zeta^5)^6$ of type $(3, \{1, 8\})$ over $\mathbb{F}_2$. Computing the product $\alpha_5 \cdot \alpha_9 = \zeta^9 \cdot (\zeta^5 + \zeta^{15} + \zeta^{40} + \zeta^{30}) = \zeta^{14} + \zeta^{24} + \zeta^4 + \zeta^{39}$ verifies that $\alpha_5 \cdot \alpha_9$ is indeed a factorization of α. $\Diamond$

PROOF (of Lemma 9.5). We subdivide the proof into three steps. Since α is normal, we have $\langle q, \mathcal{K} \rangle = \mathbb{Z}_r^\times$ by Remark 7.7.

CLAIM. *A decomposable normal Gauß period can be written as a product of a square-free Gauß period and a non-squarefree Gauß period.*

PROOF (of the claim). Let R_1 be the square-free part of r and $R_2 = \frac{r}{R_1}$, and set $a_i \equiv a \bmod R_i$ for $i = 1, 2$. For a primitive r-th root of unity ζ, we have $\zeta_i = \zeta^{r/R_i}$ a primitive R_i-th root of unity for $i = 1, 2$. Hence, $\zeta_1^a = \zeta_1^{a_1}$ and $\zeta_2^a = \zeta_2^{a_2}$. Because $\mathcal{K}$ is decomposable, we have the direct sum $\mathcal{K} = \pi_{R_1}(\mathcal{K}) \times \pi_{R_2}(\mathcal{K})$. By a straightforward computation we have:

$$\begin{aligned}
\alpha &= \sum_{a \in \mathcal{K}} b(\zeta^a) = \sum_{a \in \mathcal{K}} \zeta^{R_2 a} \cdot \prod_{1 \le i \le t, p_i | R_2} \sum_{1 \le s \le e_i} \zeta^{a R_1 R_2 / p_i^s} \\
&= \sum_{(a_1, a_2) \in \pi_{R_1}(\mathcal{K}) \times \pi_{R_2}(\mathcal{K})} (\zeta^{r/R_1})^{a_1} \cdot \prod_{1 \le i \le t, p_i | R_2} \sum_{1 \le s \le e_i} (\zeta^{r/R_2})^{a_2 R_2 / p_i^s} \\
&= \sum_{(a_1, a_2) \in \pi_{R_1}(\mathcal{K}) \times \pi_{R_2}(\mathcal{K})} b(\zeta_1^{a_1}) \cdot b(\zeta_2^{a_2}) \\
&= \sum_{a_1 \in \pi_{R_1}(\mathcal{K})} b(\zeta_1^{a_1}) \cdot \sum_{a_2 \in \pi_{R_2}(\mathcal{K})} b(\zeta_2^{a_2}).
\end{aligned}$$

The first factor is a square-free Gauß period of type $\left(\frac{\phi(R_1)}{\pi_{R_1}(\mathcal{K})}, \pi_{R_1}(\mathcal{K})\right)$ over $\mathbb{F}_q$ with respect to $\zeta_1 = \zeta^{r/R_1}$, the second one is a non-squarefree Gauß period. $\square$

CLAIM. *A decomposable non-squarefree Gauß period which is not a prime power Gauß period can be written as a product of a non-squarefree Gauß period and a prime power Gauß period.*

PROOF (of the claim). Let α be a non-squarefree Gauß period. Since it is not a prime power Gauß period, we have $t \ge 2$. Set $R = r_1 \cdots r_{t-1} \ge 2$. Then we have $r_t \ge 2$ a prime power co-prime to R. For a primitive r-th root of unity ζ, we have $\zeta_1 = \zeta^{r_t} = \zeta^{r/R}$ a primitive R-th root of unity, and $\zeta_2 = \zeta^R = \zeta^{r/r_t}$ is a

primitive r_t-th root of unity. Let $a_1 \equiv a \bmod R$ and $a_2 \equiv a \bmod r_t$. Then

$$\begin{aligned}
\alpha &= \sum_{a\in\mathcal{K}} b(\zeta^a) = \sum_{a\in\mathcal{K}} \prod_{1\le i\le t} \sum_{1\le s\le e_i} \zeta^{ar/p_i^s} \\
&= \sum_{a\in\mathcal{K}} \prod_{1\le i\le t, p_i\mid R} \sum_{1\le s\le e_i} (\zeta^{r_t})^{aR/p_i^s} \cdot \prod_{1\le i\le t, p_i\mid r_t} \sum_{1\le s\le e_i} (\zeta^{R})^{ar_t/p_i^s} \\
&= \sum_{(a_1,a_2)\in\pi_R(\mathcal{K})\times\pi_{r_t}(\mathcal{K})} \left(\prod_{1\le i\le t, p_i\mid R} \sum_{1\le s\le e_i} (\zeta_1)^{a_1 R/p_i^s} \cdot \sum_{1\le s\le e_i} (\zeta_2)^{a_2 r_t/p_i^s} \right) \\
&= \sum_{a_1\in\pi_R(\mathcal{K})} b(\zeta_1^{a_1}) \cdot \sum_{a_2\in\pi_{r_t}(\mathcal{K})} b(\zeta_2^{a_2})
\end{aligned}$$

with the first factor a non-squarefree Gauß period and the second one a prime power Gauß period. □

Claim. *A square-free Gauß period which is not a prime Gauß period can be written as a product of (conjugates of) a square-free Gauß period and a prime Gauß period.*

Proof (of the claim). Let ζ be a primitive r-th root of unity, and let R be the product $r_1 \cdots r_{t-1} > 1$ co-prime to r_t. Let $\zeta_1 = \zeta^{r_t}$ be a primitive R-th root of unity and $\zeta_2 = \zeta^R$ a primitive r_t-th root of unity. There are $u_1, u_2 \in \mathbb{Z}$ such that $u_1 r_t + u_2 R = 1$ by the Chinese Remainder Theorem. Let a_1 and a_2 be the projections of a onto $\mathbb{Z}_R^\times$ and $\mathbb{Z}_{r_t}^\times$, respectively, and set $n_1 = \frac{\phi(R)}{\#\pi_R(\mathcal{K})}$ and $n_2 = \frac{\phi(r_t)}{\#\pi_{r_t}(\mathcal{K})}$. Since α is normal, we have $\langle q, \pi_R(\mathcal{K})\rangle = \mathbb{Z}_R^\times$ and $\langle q, \pi_{r_t}(\mathcal{K})\rangle = \mathbb{Z}_{r_t}^\times$. Thus, there are $0 \le h_1 < n_1$ and $0 \le h_2 < n_2$ such that $u_1 \in q^{h_1}\pi_R(\mathcal{K})$ and $u_2 \in q^{h_2}\pi_{r_t}(\mathcal{K})$. We have

$$\begin{aligned}
\alpha &= \sum_{a\in\mathcal{K}} \zeta^a = \sum_{(a_1,a_2)\in\pi_R(\mathcal{K})\times\pi_{r_t}(\mathcal{K})} \zeta^{a_1 u_1 r_t} \cdot \zeta^{a_2 u_2 R} \\
&= \sum_{(a_1,a_2)\in\pi_R(\mathcal{K})\times\pi_{r_t}(\mathcal{K})} (\zeta_1^{a_1})^{q^{h_1}} \cdot (\zeta_2^{a_2})^{q^{h_2}} \\
&= \left(\sum_{a_1\in\pi_R(\mathcal{K})} \zeta_1^{a_1} \right)^{q^{h_1}} \cdot \left(\sum_{a_2\in\pi_{r_t}(\mathcal{K})} \zeta_2^{a_2} \right)^{q^{h_2}}
\end{aligned}$$

with the first factor a square-free Gauß period of type $(n_1, \pi_R(\mathcal{K}))$ over $\mathbb{F}_q$ with respect to $\zeta^{r/R}$, and the second factor a prime Gauß period of type $(n_2, \pi_{r_t}(\mathcal{K}))$ over $\mathbb{F}_q$ with respect to ζ^{r/r_t}. □

Induction on the number t of prime divisors of r completes the proof of the lemma. □

9.2. Fast multiplication for decomposable Gauß periods. If a normal Gauß period is decomposable then its factorization into prime and prime power Gauß periods is related to a tower of fields. Each Gauß period along this tower satisfies the assumptions of Fact 7.21, i.e. the extension degrees are pairwise co-prime.

PROPOSITION 9.6. *Let r, q, n, k be positive integers such that $q \geq 2$ and $r \geq 2$ are co-prime and $\phi(r) = nk$. Let $r_1 \cdots r_t$ be the prime power decomposition of r. Let $\mathcal{K}$ be a subgroup of $\mathbb{Z}_r^\times$ of order k, set $\mathcal{K}_i = \pi_{r_i}(\mathcal{K})$ its image of order k_i onto $\mathbb{Z}_{r_i}^\times$ under the canonical projection π_{r_i}, and $n_i = \frac{\phi(r_i)}{k_i}$ for $1 \leq i \leq t$. Then the following are equivalent:*

(i) $\langle q, \mathcal{K} \rangle = \mathbb{Z}_r^\times$ *and* $\mathcal{K}$ *is decomposable.*

(ii) $\langle q, \mathcal{K}_i \rangle = \mathbb{Z}_{r_i}^\times$ *for all* $1 \leq i \leq t$, *and* $n = n_1 \cdots n_t$ *with* $n_1, \ldots, n_t$ *pairwise co-prime.*

PROOF. "(i)⇒(ii)" The canonical projection π_{r_i} is an epimorphism. Thus, $\mathbb{Z}_{r_i}^\times = \pi_{r_i}(\mathbb{Z}_r^\times) = \pi_{r_i}(\langle q, \mathcal{K} \rangle) = \langle q, \mathcal{K}_i \rangle$ for all $1 \leq i \leq t$. Since $\mathcal{K}$ is decomposable, we have $k = k_1 \cdots k_t$ and $n = \frac{\phi(r)}{k} = \prod_{1 \leq i \leq t} \frac{\phi(r_i)}{k_i} = \prod_{1 \leq i \leq t} n_i$. We prove by induction on the number of prime divisors that $n_1, \ldots, n_t$ are pairwise co-prime. For $i = 1$ there is nothing to show. Thus, we suppose that the claim is true for $\mathcal{K}' = \mathcal{K}_1 \times \cdots \times \mathcal{K}_i$ which is a decomposable subgroup of $\mathbb{Z}_{r'}^\times$ of order k' where $r' = r_1 \cdots r_i$. By construction we have $\langle q, \mathcal{K}' \rangle = \mathbb{Z}_{r'}^\times$ and $n' = \frac{\phi(r')}{k'} = n_1 \cdots n_i$. We suppose that $d = \gcd(n', n_{i+1}) > 1$, i.e. $n' \cdot \frac{n_{i+1}}{d} < n_1 \cdots n_{i+1}$. Since $q^{n_{i+1}} \in \mathcal{K}_{i+1}$ by Fact 7.6(iii), we have $q^{n_{i+1} \cdot n'/d} \in \mathcal{K}_{i+1}$. But also $q^{n' \cdot n_{i+1}/d} \in \mathcal{K}'$ since $q^{n'} \in \mathcal{K}'$, and we conclude with the help of the Chinese Remainder Theorem that $q^{n' \cdot n_{i+1}/d} \in \mathcal{K}' \times \mathcal{K}_{i+1}$. Then $\#\langle q, \mathcal{K}' \times \mathcal{K}_{i+1} \rangle \leq \frac{n' \cdot n_{i+1}}{d} \cdot k' \cdot k_{i+1} < (n' \cdot k') \cdot (n_{i+1} \cdot k_{i+1}) = \phi(r') \cdot \phi(r_{i+1}) = \#(\mathbb{Z}_{r_1}^\times \times \cdots \times \mathbb{Z}_{r_{i+1}}^\times)$ which is a contradiction. Hence, n' and n_{i+1} are co-prime. The induction hypothesis guarantees that $n_1, \ldots, n_i$ are pairwise co-prime, and the claim holds for $n_1, \ldots, n_{i+1}$.

"(ii)⇒(i)" The group $\mathcal{K}$ can be regarded as a subgroup of $\mathcal{K}_1 \times \cdots \times \mathcal{K}_t$; hence k is a divisor of $k_1 \cdots k_t$. By assumption we have $n = n_1 \cdots n_t$. Thus, $k = \frac{\phi(r)}{n} = \prod_{1 \leq i \leq t} \frac{\phi(r_i)}{n_i} = k_1 \cdots k_t$, i.e. the subgroup $\mathcal{K}$ is decomposable. We always have $\langle q, \mathcal{K} \rangle \subseteq \mathbb{Z}_r^\times$, and it remains to prove the other inclusion to show equality. Let a be an element in $\mathbb{Z}_r^\times$ and $a_i = \pi_{r_i}(a)$ for all $1 \leq i \leq t$. For $\langle q, \mathcal{K}_i \rangle = \mathbb{Z}_{r_i}^\times$ there are $c_i' \in \mathcal{K}_i$ and $0 \leq h_i < n_i$ such that $a_i = q^{h_i} c_i'$ for $1 \leq i \leq t$. But $n_1, \ldots, n_t$ are pairwise co-prime, and by the Chinese Remainder Theorem exist $0 \leq h < n$ with $h \equiv h_i \bmod n_i$ for $1 \leq i \leq t$. Since $q^{n_i} \in \mathcal{K}_i$, we have $q^h \equiv q^{h_i} c_i'' \bmod r_i$ for suitable $c_i'' \in \mathcal{K}_i$, $1 \leq i \leq t$.

We set $c = (c'_1/c''_1, \ldots, c'_t/c''_t) \in \mathcal{K}$ to get $a \equiv q^h c \bmod r$. Thus, $\langle q, \mathcal{K}\rangle \supseteq \mathbb{Z}_r^\times$, i.e. $\langle q, \mathcal{K}\rangle = \mathbb{Z}_r^\times$ as claimed. □

The factorization of a normal decomposable Gauß period α offers a recursive approach to do multiplication fast whenever $\mathbb{F}_{q^n}$ is represented by the normal basis $\mathcal{N} = (\alpha, \ldots, \alpha^{q^{n-1}})$.

REMARK 9.7. *Let n_1 and n_2 be two co-prime integers, and set $n = n_1 \cdot n_2$. Let $\alpha_1 \in \mathbb{F}_{q^{n_1}}$ and $\alpha_2 \in \mathbb{F}_{q^{n_2}}$ be normal elements over $\mathbb{F}_q$, and $\alpha = \alpha_1 \cdot \alpha_2$ be a normal element in $\mathbb{F}_{q^n}$.*

(i) The element α_2 is normal in $\mathbb{F}_{q^n}$ over $\mathbb{F}_{q^{n_1}}$.

(ii) Transforming an element given as linear combination of the conjugates of α over $\mathbb{F}_q$ into a linear combination of the conjugates of α_2 over $\mathbb{F}_{q^{n_1}}$ can be computed without operations in $\mathbb{F}_q$.

PROOF. (i) This is just Lemma 7.25(ii).

(ii) Let $A = \sum_{0 \le h < n} A_h \alpha^{q^h}$ be an element in $\mathbb{F}_{q^n}$. Let $h_i \equiv h \bmod n_i$ for $i = 1, 2$. Then $\alpha^{q^h} = \alpha_1^{q^{h_1}} \cdot \alpha_2^{q^{h_2}}$ and

$$A = \sum_{0 \le h < n_1 n_2} A_h \left(\alpha_1^{q^{h_1}} \cdot \alpha_2^{q^{h_2}}\right) = \sum_{0 \le h_2 < n_2} \left(\sum_{0 \le h_1 < n_1} A_{(h_1,h_2)} \alpha_1^{q^{h_1}}\right) \alpha_2^{q^{h_2}}$$

where we identify h and $(h_1, h_2) = (h \bmod n_1, h \bmod n_2)$. Since n_1 and n_2 are co-prime, we have $\{n_1 a \operatorname{rem} n_2 \colon 0 \le a < n_2\} = \{0 \le a < n_2\}$ and

$$A = \sum_{0 \le h_2 < n_2} \left(\sum_{0 \le h_1 < n_1} A_{h_1, n_1 h_2} \alpha_1^{q^{h_1}}\right) \alpha_2^{(q^{n_1})^{h_2}}.$$

This is just a resorting of the coefficients of A which can be done without operations in $\mathbb{F}_q$. □

A constructive proof. We are now ready to give a constructive proof how to apply fast polynomial multiplication if $\mathbb{F}_{q^n}$ is given by a normal basis $\mathcal{N} = (\alpha, \ldots, \alpha^{q^{n-1}})$ over $\mathbb{F}_q$, and α is a decomposable Gauß period.

THEOREM 9.8. *Let α be a decomposable normal Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ with $\mathcal{K}$ a subgroup of $\mathbb{Z}_r^\times$, and let $r_1 \cdots r_t$ be the prime power decomposition of r. Then two elements in $\mathbb{F}_{q^n}$ given as linear combinations of the elements of the normal basis $\mathcal{N} = (\alpha, \ldots, \alpha^{q^{n-1}})$ can be multiplied with at most $O(\prod_{1 \le i \le t} \mathsf{M}(r_i))$ operations in $\mathbb{F}_q$.*

PROOF. We prove by induction on the number of prime divisors t of r. If $t = 1$ the claim follows with Theorem 8.26. Now we suppose $t \geq 2$. We can write $\alpha = \prod_{1 \leq i \leq t} \alpha_i^{q^{h_i}}$ as a product of conjugates of normal prime and prime power Gauß periods α_i of type $(n_i, \pi_{r_i}(\mathcal{K}))$ over $\mathbb{F}_q$ by Lemma 9.5. Set $n' = \frac{n}{n_t}$. The element $\alpha' = \prod_{1 \leq i \leq t-1} \alpha_i^{q^{h_i}}$ is normal in $\mathbb{F}_{q^{n'}}$ over $\mathbb{F}_q$. Since α is decomposable, Proposition 9.6 claims that n' and n_t are co-prime. Then α_t is a normal prime or prime power Gauß period in $\mathbb{F}_{q^n}$ over $\mathbb{F}_{q^{n'}}$ by Remark 9.7(i). As claimed in Remark 9.7(ii), we can multiply two elements in $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ by multiplying them in $\mathbb{F}_{q^n}$ over $\mathbb{F}_{q^{n'}}$. By Theorem 8.26, the multiplication can be done with at most $O(\mathsf{M}(r_t))$ operations (additions, multiplications) in $\mathbb{F}_{q^{n'}}$. Moreover, α' is a decomposable normal Gauß period of type $(n', \pi_{r_1}(\mathcal{K}) \times \cdots \times \pi_{r_{t-1}}(\mathcal{K}))$ over $\mathbb{F}_q$. By the induction hypothesis, multiplication in $\mathbb{F}_{q^{n'}}$ can be done with at most $O(\prod_{1 \leq i \leq t-1} \mathsf{M}(r_i))$ operations in $\mathbb{F}_q$, and the claim follows. □

EXAMPLE 9.3 CONTINUED. The decomposable Gauß period $\alpha = \zeta^{14} + \zeta^{24} + \zeta^4 + \zeta^{39}$ of type $(12, \{1, 26\})$ with $\{1, 26\} \subset \mathbb{Z}_{45}^\times$ over $\mathbb{F}_2$ is normal in $\mathbb{F}_{2^{12}}$. We calculate the product $\alpha^{2^2} \cdot \alpha$.

(i) As shown above, α factors into $\alpha = \alpha_5 \cdot \alpha_9$ with α_5 a prime Gauß period of type $(4, 1)$ over $\mathbb{F}_2$, and α_9 a prime power Gauß period of type $(3, \{1, 8\})$ over $\mathbb{F}_2$ where $\{1, 8\} \subset \mathbb{Z}_9^\times$. We transform the task into a multiplication over $\mathbb{F}_8$:

$$\alpha^4 \cdot \alpha = (\alpha_5^4 \cdot \alpha_9^4) \cdot (\alpha_5 \cdot \alpha_9) = (\alpha_5^4 \cdot \alpha_5) \cdot (\alpha_9^4 \cdot \alpha_9).$$

Now $\alpha_9^4 \cdot \alpha_9 = \alpha_9^2 + \alpha_9^4$ as computed in Example 8.2.

(ii) It remains to perform the arithmetic in $\mathbb{F}_8$ over $\mathbb{F}_2$. Since α_5 is a prime Gauß period, we have

$$\alpha_5^4 \cdot \alpha_5 = (\zeta^9)^4 \cdot (\zeta^9) = (\zeta^9)^5 = 1 = \alpha_5 + \alpha_5^2 + \alpha_5^4 + \alpha_5^8.$$

(iii) Combining both results gives

$$\begin{aligned} \alpha^4 \cdot \alpha &= (\alpha_5 + \alpha_5^2 + \alpha_5^4 + \alpha_5^8) \cdot (\alpha_9^2 + \alpha_9^4) \\ &= \alpha_5^{2^0} \alpha_9^{2^1} + \alpha_5^{2^1} \alpha_9^{2^1} + \alpha_5^{2^2} \alpha_9^{2^1} + \alpha_5^{2^3} \alpha_9^{2^1} + \alpha_5^{2^0} \alpha_9^{2^2} + \alpha_5^{2^1} \alpha_9^{2^2} \\ &\quad + \alpha_5^{2^2} \alpha_9^{2^2} + \alpha_5^{2^3} \alpha_9^{2^2} \\ &= \alpha^{2^4} + \alpha^{2^1} + \alpha^{2^{10}} + \alpha^{2^7} + \alpha^{2^8} + \alpha^{2^5} + \alpha^{2^2} + \alpha^{2^{11}}, \end{aligned}$$

since $\alpha^{2^h} = \alpha_5^{2^{h_1}} \cdot \alpha_9^{2^{h_2}} = (\alpha_5 \cdot \alpha_9)^{2^{9h_1 + 4h_2}}$. ◊

Experiments. We composed another test series *squarefreeGP* to illustrate the arithmetic of decomposable normal Gauß periods. This test series contained a special type of decomposable Gauß periods; *square-free Gauß periods* play an important rôle within general Gauß periods. Square-free means that all factors of the prime power decomposition of r are different primes. Square-free Gauß periods preserve the classical form given by prime Gauß periods: $\alpha = \sum_{a \in \mathcal{K}} \zeta^a$ for a subgroup $\mathcal{K} \subseteq \mathbb{Z}_r^\times$. They were discussed by von zur Gathen & Schlink (1996) as a first step towards a generalization of prime Gauß periods. From our algorithmic point of view, square-free Gauß periods are the simplest non-trivial decomposable Gauß periods. Table 9.1 shows the significance of this type for fast arithmetic. Square-free Gauß periods are very useful to get smaller values for k compared to the other three types including prime Gauß periods. In particular, for extension

Minimal value of the parameter k for normal Gauß periods with respect to the class								
class \ q	2	3	5	7	11	13	17	19
prime	57.79	63.04	63.25	63.24	64.71	65.27	64.93	65.20
squarefree	26.19	29.22	30.35	23.35	25.78	25.16	32.33	22.59
prime power	0.87	0.89	0.92	0.84	0.95	1.08	0.79	0.62
general	2.66	6.85	5.48	12.56	8.56	8.49	1.95	11.58
no normal GP	12.49	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 9.1: The percentage for which the minimal parameter $k \in \mathbb{N}_{\geq 1}$ is given by the special class of a Gauß period. The values are given for all field extensions $\mathbb{F}_{q^n}$ with $2 \leq n < 10000$. The values for q are given in the first row; e.g. the distribution over the binary field $\mathbb{F}_2$ are listed in the second column. The search for $k = \phi(r)/n$ is restricted to $r \leq 1000000$. If the minimal k is given for more than one class of Gauß period then it is attached to only one; the chosen class is then with respect to the rows from top to bottom.

fields over $\mathbb{F}_2$ nearly all smallest values for $k \in \mathbb{N}_{\geq 1}$ are given by either prime or square-free Gauß periods. There are only 2.66% of all extensions $\mathbb{F}_{2^n}$, $2 \leq n < 10000$, for which the minimal k is not given by a prime, a prime power or a square-free Gauß period. A comparison between Table 9.2 and Table 9.4 supports the importance of square-free Gauß periods. We implemented the algorithm that is included in the proof of Theorem 9.8 for square-free Gauß periods. Our implementation based on the fast polynomial arithmetic of the software package BiPolAr. Since BiPolAr restricts its powerful routines to polynomials over the binary field $\mathbb{F}_2$, we had to modify the arithmetic along the tower of fields. One possible, but not realized, solution would be to apply *Kronecker substitution*, see von zur Gathen & Gerhard (1999), Section 8.4. The price to pay is another blow-up of the degree of the polynomials. Therefore, we have chosen another

Existence of normal bases generated by a square-free Gauß period with given parameter $k \in \mathbb{N}_{\geq 1}$								
$k \quad \backslash \quad q$	2	3	5	7	11	13	17	19
$k = 1$	4.70	4.76	4.92	4.65	4.43	4.57	4.50	4.72
$k \leq 2$	25.22	25.78	24.60	23.21	23.77	22.67	25.18	22.75
$k \leq \log_2 n$	75.90	86.23	86.11	85.18	85.24	84.51	86.31	83.84
$k \leq \sqrt{n}$	87.24	99.65	99.68	99.63	99.66	99.57	99.57	99.50
$k < \infty$	87.51	100.00	100.00	100.00	100.00	100.00	100.00	99.98
theo.	87.51	100.00	100.00	100.00	100.00	100.00	100.00	100.00

Table 9.2: Percentage of field extensions $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ with $2 \leq n < 10000$ for which there is a normal basis given by a square-free Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$. The rows show the distribution if the value for $k = \#\mathcal{K}$ is also restricted. We limited our experiments for r with $\phi(r) = nk$ to $2 \leq r < 1000000$. The last row labeled *theo.* is due to Fact 9.18.

way and used the trace map directly to identify the elements of the intermediate fields.

For the test series *squarefreeGP*, we chose 50 values $n \approx 200i$ for $1 \leq i \leq 50$. Our selection was restricted to extension fields $\mathbb{F}_{2^n}$ for which there is a normal square-free Gauß period of type $(n, \mathcal{K})$ with $k = \#\mathcal{K} \leq 4$. We compared these square-free Gauß periods with prime Gauß periods. In particular, we wanted to examine the significance of the parameter k. Thus, for all chosen n of the test series *squarefreeGP* there is a normal prime Gauß period, and the minimal k of this prime Gauß period is close to that of the square-free Gauß period. For most values the square-free Gauß period offers a smaller value of k. But we also chose 9 values for n where both k are equal, and for 6 values the parameter k of the prime Gauß period is smaller than that for the square-free Gauß period. The details are listed in Table A.18, columns 2 and 7.

As for the test series *PrimepowerGP*, the direct comparison confirms that the parameter k is in general a suitable indicator. As expected, the prime Gauß period performed faster multiplication whenever its parameter k was at most that of the square-free Gauß period. Otherwise, square-free Gauß periods were mostly superior if they had a smaller k. The only counterexample in our test series was given for $n = 4202$. Here the square-free Gauß period has parameter $k = 4$, and the prime Gauß period is of type $(4202, 5)$ over $\mathbb{F}_2$. But the maximal degree of the polynomials is marked by the integer r with $\phi(r) = nk$, see columns 3 and 8 of Table A.18. In the case $n = 4202$ the degree is smaller for the prime Gauß period even though the parameter k is larger. Two further examples were given by $n = 8622$ and $n = 8782$. Here the parameters k differ for both classes of Gauß periods but the values for r are nearly equal. But whenever both the parameter k and r decreased then our algorithm for general Gauß periods beat the prime

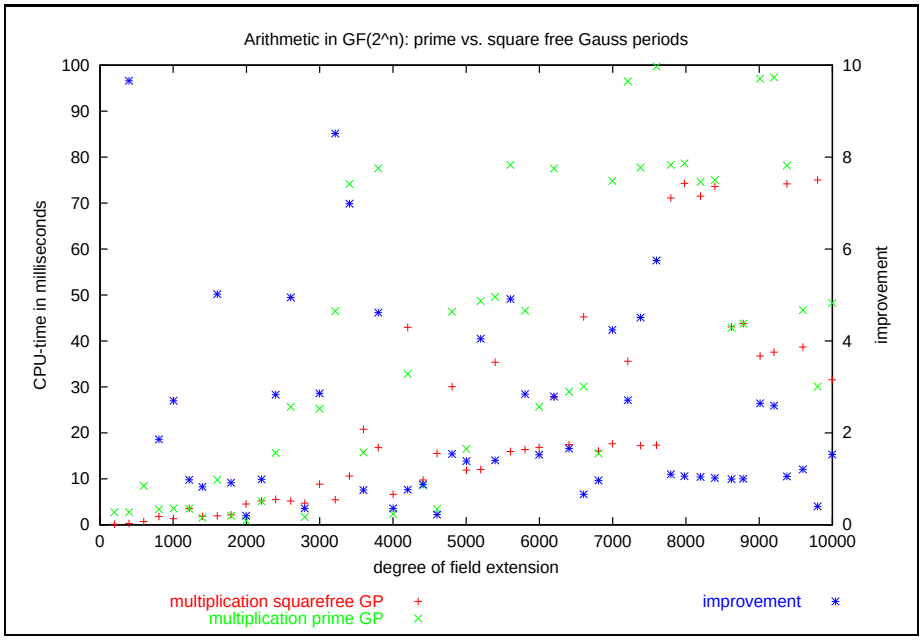


Figure 9.1: Times for multiplication for the test series *squarefreeGP*. Square-free Gauß periods of type $(n, \mathcal{K})$ are marked by $+$, prime Gauß periods of type (n, k) by $\times$. The times are the average of 10000 trials. They are related to the left y-axis. The scale of the right y-axis marks the quotient $(*)$ between the order of the subgroup $\mathcal{K}$ for the square-free Gauß period and the order k of the subgroup for the prime Gauß period.

case, see Table A.18 and Figure 9.1. The highest speed-up for the test series *squarefreeGP* was given for $n = 198$; here the speed-up is 21.73. For most values of n where k is larger for the prime Gauß period the improvement is between 1.5 and 6. On average over all values of the test series *squarefreeGP*, replacing a prime by a square-free Gauß period decreased the value for r by 1.86 and sped up the multiplication time by 2.95. Thus, our experiments show that a suitable decrease for k, given by the substitution of a normal prime Gauß period by a normal general Gauß period, results in faster arithmetic for $\mathbb{F}_{q^n}$.

9.3. Existence of decomposable Gauß periods. There is one step missing to derive Result 9.1 from Theorem 9.8: Not every normal Gauß period is decomposable as already illustrated in Example 9.3. But a normal Gauß period in $\mathbb{F}_{q^n}$ always indicates a decomposable normal Gauß period in $\mathbb{F}_{q^n}$!

COROLLARY 9.9. *Let r, q, n, k be positive integers with $r, q \geq 2$ such that r and q are co-prime and $\phi(r) = nk$. The following are equivalent:*

(i) There is a normal Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ with $\mathcal{K}$ a subgroup of $\mathbb{Z}_r^\times$ of order k.

(ii) There is a decomposable normal Gauß period of type $(n, \mathcal{L})$ *over* $\mathbb{F}_q$ *with* $\mathcal{L}$ *a decomposable subgroup of* $\mathbb{Z}_r^\times$ *of order* k.

Gao's lemma. The proof of Corollary 9.9 is based on a result of Gao (2001) which we discuss primarily.

GAO'S LEMMA 9.10 (Gao 2001, Theorem 1.1). *Let* $\mathcal{Z}$ *be an Abelian group of finite order. Let* $\mathcal{Q}$ *be a subset and* $\mathcal{K}$ *be a subgroup of* $\mathcal{Z}$ *such that* $\mathcal{Z} = \langle \mathcal{Q}, \mathcal{K} \rangle$. *Then, for any direct sum of* $\mathcal{Z} = \mathcal{Z}_1 \times \cdots \times \mathcal{Z}_t$, *there exists a subgroup* $\mathcal{L}$ *of the form* $\mathcal{L} = \mathcal{L}_1 \times \cdots \times \mathcal{L}_t$ *with* $\mathcal{L}_i$ *a normal subgroup of* $\mathcal{Z}_i$ *for* $1 \le i \le t$ *such that* $\mathcal{Z} = \langle \mathcal{Q}, \mathcal{L} \rangle$ *and* $\mathcal{Z}/\mathcal{L} \cong \mathcal{Z}/\mathcal{K}$.

Translating this result into our special (and somehow simpler) situation, we get the following version.

COROLLARY 9.11. *Let* r *and* q *be co-prime positive integers greater than* 2, *and* $r_1 \cdots r_t$ *be the prime power decomposition of* r. *If there is a subgroup* $\mathcal{K}$ *of* $\mathbb{Z}_r^\times$ *with* $\langle q, \mathcal{K} \rangle = \mathbb{Z}_r^\times$ *then there is a decomposable subgroup* $\mathcal{L}$ *of* $\mathbb{Z}_r^\times$ *of same order* $\#\mathcal{L} = \#\mathcal{K}$ *such that* $\langle q, \mathcal{L} \rangle = \mathbb{Z}_r^\times$.

We prove the latter version by giving a construction for a decomposable subgroup $\mathcal{L}$. We start with some preliminaries.

The group of units modulo a prime power. We recall the basic structure of $\mathbb{Z}_r^\times$ for a prime power r.

FACT 9.12 (Hasse 1969, p. 57). *Let* r *be a prime power.*

(i) If r *is not divisible by* 8 *then* $\mathbb{Z}_r^\times$ *is a cyclic group.*

(ii) If 8 *divides* r, *i.e.* $r = 2^e$ *with* $e \ge 3$, *then* $\mathbb{Z}_{2^e}^\times$ *is the direct product of the two cyclic groups* $\pm 1 = \langle -1 \bmod 2^e \rangle$ *and* $\mathcal{Z}_{2^e} = \langle 5 \bmod 2^e \rangle$.

We first turn to cyclic groups $\mathbb{Z}_r^\times$, i.e. r is not divisible by 8.

REMARK 9.13. *Let* r *and* q *be two co-prime prime powers,* $r \not\equiv 0 \bmod 8$, *and* $\mathcal{K}$ *a subgroup of* $\mathbb{Z}_r^\times$ *of order* $k \ge 1$ *such that* $\phi(r) = nk$ *and* $\langle q, \mathcal{K} \rangle = \mathbb{Z}_r^\times$. *Let* p *be a prime dividing* $n = \frac{\phi(r)}{k}$, *and set* $e = \max\{e' \in \mathbb{N}_{\ge 1} : p^{e'} \text{ divides } \phi(r)\} \ge 1$. *Then* $\langle q \rangle$ *contains the uniquely determined subgroup*[61] *of order* p^e *of* $\mathbb{Z}_r^\times$.

PROOF. Since $r \not\equiv 0 \bmod 8$, we have $\mathbb{Z}_r^\times$ a cyclic group. Let $\xi \in \mathbb{Z}_r^\times$ be a generator of the multiplicative group modulo r, i.e. $\langle \xi \rangle = \mathbb{Z}_r^\times$. Then there is a uniquely determined subgroup $\mathcal{H}$ of order p^e in $\mathbb{Z}_r^\times$; it is generated by $\xi^{\phi(r)/p^e}$. Since $e \in \mathbb{N}_{\ge 1}$ is maximal, we have $\gcd(\frac{\phi(r)}{p^e}, p) = 1$.

[61] Such a subgroup is also called *p-(Sylow) group* (see e.g. Jacobson (1974), p. 78).

We first suppose that neither $N = \mathrm{ord}_r(q)$ nor $k = \#\mathcal{K}$ are divisible by p^e. All elements in $\langle q, \mathcal{K}\rangle$ are of the type $\xi^{a\frac{\phi(r)}{N}+b\frac{\phi(r)}{k}}$, $a, b \in \mathbb{Z}$. Because e is maximal such that p^e divides $\phi(r)$ and p^e does not divide N or k, no element has order p^e. Thus, $\mathcal{H}$ is not a subgroup of $\langle q, \mathcal{K}\rangle$, which contradicts $\mathcal{H} \subseteq \mathbb{Z}_r^\times = \langle q, \mathcal{K}\rangle$. Therefore, we have N or k a multiple of p^e.

Now, we suppose that p^e divides k, i.e. $\mathcal{H} \subseteq \mathcal{K}$. Since e is maximal and $\phi(r) = nk$, we know that $n = \frac{\phi(r)}{k}$ is not divisible by p. This is again a contradiction. Thus, p^e divides N, and we conclude that $\mathcal{H}$ is a subgroup of $\langle q\rangle$. □

Our basic idea to prove Corollary 9.11 is to manipulate the subgroup $\mathcal{K} \subseteq \mathbb{Z}_r^\times$ by changing the corresponding subgroups $\mathcal{L}_i$ in $\mathbb{Z}_{r_i}^\times$ for all factors $r_1, \ldots, r_t$ of the prime power decomposition. These manipulations have to satisfy the invariant that $\langle q, \mathcal{K}\rangle = \mathbb{Z}_r^\times$ at any time. In what follows, we set $e = \max\{e' \in \mathbb{N}_{\geq 1} : p^{e'} \text{ divides } \phi(r)\}$ and $\mathcal{H}$ the uniquely determined subgroup of order p^e of the cyclic group $\mathbb{Z}_r^\times$. By $N = \mathrm{ord}_r(q)$ we denote the order of q in $\mathbb{Z}_r^\times$. We assume p to be a divisor of $n = \frac{\phi(r)}{k}$.

REMARK 9.14. *Let N, n, k, p, q, and e be as above. If $\langle q, \mathcal{K}\rangle = \mathbb{Z}_{p^e}^\times$ then*

$$\mathcal{L} = \langle q^{N/p^e}, \mathcal{K}\rangle$$

is a subgroup of $\mathbb{Z}_{p^e}^\times$ of order $\mathrm{lcm}(p^e, k)$ satisfying $\langle q, \mathcal{L}\rangle = \mathbb{Z}_{p^e}^\times$.

PROOF. Since p divides n, we have $\mathcal{H} \subseteq \langle q\rangle$ by Remark 9.13, and $\mathcal{H}$ is generated by q^{N/p^e}. We set $\mathcal{L} = \langle \mathcal{H}, \mathcal{K}\rangle$ the subgroup of order $\frac{p^e \cdot k}{\gcd(p^e,k)} = \mathrm{lcm}(p^e, k)$. Then $\langle q, \mathcal{L}\rangle = \langle q, \mathcal{H}, \mathcal{K}\rangle = \langle q, \mathcal{K}\rangle = \mathbb{Z}_r^\times$ as claimed. □

Since e is maximal, we know that $n' = \frac{\phi(r)}{\#\mathcal{L}} = \frac{\phi(r)\cdot\gcd(p^e,k)}{p^e\cdot k}$ is not divisible by p^e. All other prime divisors of $n = \frac{\phi(r)}{\#\mathcal{K}}$ are prime divisors of n'. Thus, this tool enables us to erase successively all common divisors of the $n_i = \frac{\phi(r_i)}{\#\pi_{r_i}(\mathcal{K})}$ for $1 \leq i \leq t$. Another tool is used to delete all elements of a given order in $\mathcal{K}$.

REMARK 9.15. *Let N, n, k, p, q, and e be as above, p a divisor of n, and $0 \leq f < e$. If $\langle q, \mathcal{K}\rangle = \mathbb{Z}_{p^e}^\times$ then*

$$\mathcal{L} = \langle q^{N/p^{e-f}}, \{a^{p^e} : a \in \mathcal{K}\}\rangle$$

is a subgroup of $\mathbb{Z}_{p^e}^\times$ of order $\frac{k\cdot p^{e-f}}{\gcd(k,p^e)}$ satisfying $\langle q, \mathcal{L}\rangle = \mathbb{Z}_{p^e}^\times$.

PROOF. We can write $\mathcal{K} = \{(\xi^{\phi(r)/k})^i : 0 \leq i < k\}$, since ξ is a generator of $\mathbb{Z}_r^\times$. Then $\mathcal{L}' = \{a^{p^e} : a \in \mathcal{K}\} = \{(\xi^{\phi(r)/k})^{ip^e} : 0 \leq i < k\}$ is a subgroup of order $k' = \frac{k}{\gcd(p^e,k)}$. By Remark 9.13, the group $\mathcal{H}$ of order p^e is a subgroup of $\langle q\rangle$. We conclude that $\mathbb{Z}_r^\times = \langle q, \mathcal{K}\rangle = \langle q, \mathcal{L}'\rangle$. We add all elements of the p^f-subgroup $\langle q^{N/p^{e-f}}\rangle$ of $\mathbb{Z}_r^\times$ to $\mathcal{L}'$. This generates the subgroup $\mathcal{L} = \langle q^{N/p^{e-f}}, \mathcal{L}'\rangle$ which has order $\mathrm{lcm}(p^{e-f}, k') = \frac{p^{e-f}\cdot k}{\gcd(p^e,k)}$ and satisfies $\langle q, \mathcal{L}\rangle = \langle q, \mathcal{L}'\rangle = \mathbb{Z}_{p^e}^\times$. □

Applying this tool, $n' = \frac{\phi(r)}{\#\mathcal{L}}$ is divisible by p^f but not by p^{f+1}. As above, this transformation has no effect on the other prime divisors of $n = \frac{\phi(r)}{k}$.

Let $r_1 \cdots r_t$ be the prime power decomposition of r. For the cyclic subgroups $\mathbb{Z}_{r_i}^\times$, $1 \le i \le t$, we are now ready to assign suitable prime divisors of n to r_i and erase others. It remains to get the tools for $\mathbb{Z}_{2^e}^\times$ for $e \ge 3$. For $r = 2^e$ with $e \ge 3$ the group $\mathbb{Z}_{2^e}^\times$ is no longer cyclic, but the necessary tool is simpler here. We suppose that $\langle q, \mathcal{K}\rangle = \mathbb{Z}_{2^e}^\times$ for a subgroup $\mathcal{K}$ of order k. Then both k and $n = \frac{\phi(2^e)}{k} = \frac{2^{e-1}}{k}$ are powers of 2. Thus, it is easy to find a subgroup $\mathcal{L} \subseteq \mathbb{Z}_{2^e}^\times$ in the two cases we are interested in. For $n' = \frac{\phi(2^e)}{\#\mathcal{L}} = 1$ we set $\mathcal{L} = \mathbb{Z}_{2^e}^\times$, and for $n' = n$ we have $\mathcal{L} = \mathcal{K}$. Both choices satisfy $\langle q, \mathcal{L}\rangle = \mathbb{Z}_{2^e}^\times$.

Finally, we cite the following fact which is a direct consequence of the Chinese Remainder Theorem.

FACT 9.16. *Let r be a positive integer and $r_1 \cdots r_t$ its prime power decomposition, and let q be an integer greater than 1 such that* $\gcd(r, q) = 1$. *Then* $N = \mathrm{ord}_r(q) = \mathrm{lcm}(\mathrm{ord}_{r_1}(q), \ldots, \mathrm{ord}_{r_t}(q))$.

A proof of Gao's lemma. We are now ready to give a proof of our tailor-made version of Gao's lemma.

PROOF (of Corollary 9.11). Let $r_1 \cdots r_t$ and $n'_1 \cdots n'_s$ be the prime power decompositions of r and n, respectively. By assumption, we have $\langle q, \mathcal{K}\rangle = \mathbb{Z}_r^\times$. Thus, $\langle q, \mathcal{L}_j\rangle = \mathbb{Z}_{r_j}^\times$ for all $\mathcal{L}_j = \pi_{r_j}(\mathcal{K})$ with $1 \le j \le t$. We use the tools prepared above to assign $n' = p^f \in \{n'_1, \ldots, n'_s\}$ to exactly one prime divisor of r. The manipulation has no effect on the invariant $\langle q, \mathcal{L}_j\rangle = \mathbb{Z}_{r_j}^\times$ and on the divisors of $n_j = \frac{\phi(r_j)}{\#\mathcal{L}_j}$ for $1 \le j \le t$ which are co-prime to n'.

Since $\langle q, \mathcal{K}\rangle = \mathbb{Z}_r^\times$, we have that n' divides $N = \mathrm{ord}_r(q)$. By Fact 9.16, there is an $i \in \{1, \ldots, t\}$ such that n' divides $N_i = \mathrm{ord}_{r_i}(q)$. We distinguish between two cases:

- Let $n' = p^f$ be not a power of two. Then r_i is not a power of 2 and $\mathbb{Z}_{r_i}^\times$ is cyclic. Let $e = \max\{e' \in \mathbb{N}_{\ge 1} : p^{e'} \text{ divides } \phi(r_i)\}$. We set the modified subgroup $\mathcal{L}_i$ to be $\langle q^{N_i/p^{e-f}}, \{a^{p^e} : a \in \mathcal{L}_i\}\rangle$. By Remark 9.15, this is a subgroup that satisfies $\langle q, \mathcal{L}_i\rangle = \mathbb{Z}_{r_i}^\times$ and $\gcd(\frac{\phi(r_i)}{\#\mathcal{L}_i}, p^e) = n'$ as claimed. For all other $1 \le j \le t$ with $j \ne i$ we leave $\mathcal{L}_j$ unchanged if p does not divide $n_j = \frac{\phi(r_j)}{\#\mathcal{L}_j}$. Otherwise, we modify $\mathcal{L}_j$ according to Remark 9.14.

- If n' is a power of two then there are two cases. If there exists $1 \le i \le t$ such that n' divides $N_i = \mathrm{ord}_{r_i}(q)$ and $\mathbb{Z}_{r_i}^\times$ is cyclic, then we proceed as in the cyclic case using Remark 9.14 for r_i. For $1 \le j \le t$, $i \ne j$, we apply Remark 9.15 for all odd r_j. If here is also a $1 \le j \le t$ such that r_j is a power of 2 then we set $\mathcal{L}_j = \mathbb{Z}_{r_j}^\times$. Otherwise, no odd prime divisor has a

subgroup of order n'. Then, we have $r_i = 2^{e'}$ with $e' \geq 3$. By Fact 9.16, we have $n' = \frac{\phi(r_i)}{\#\mathcal{L}_i}$. Thus, we do not change $\mathcal{L}_i$. For $1 \leq j \leq t$ with $j \neq i$ and $\frac{\phi(r_j)}{\#\mathcal{L}_j}$ even, we use Remark 9.14 again.

By the steps described above, we can successively construct subgroups $\mathcal{L}_1, \ldots, \mathcal{L}_t$ such that $n = n_1 \cdots n_t$ and all $n_i = \frac{\phi(r_i)}{\#\mathcal{L}_i}$, $1 \leq i \leq t$, are pairwise co-prime. Moreover, $\langle q, \mathcal{L}_i \rangle = \mathbb{Z}_{r_i}^{\times}$ for all $1 \leq i \leq t$. By Proposition 9.6, this generates a decomposable subgroup $\mathcal{L} = \mathcal{L}_1 \times \cdots \times \mathcal{L}_t$ of order $k = \#\mathcal{K}$ which satisfies $\langle q, \mathcal{L} \rangle = \mathbb{Z}_r^{\times}$ as claimed. □

Normal decomposable Gauß periods. We can derive the Normal Gauß period theorem 7.5 for decomposable Gauß periods from Theorem 8.41 with the help of Proposition 9.6.

THEOREM 9.17. *Let α be a decomposable Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ with $\mathcal{K}$ a subgroup of $\mathbb{Z}_r^{\times}$. Then α is normal in $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ if and only if $\langle q, \mathcal{K} \rangle = \mathbb{Z}_r^{\times}$.*

PROOF. Let α be a decomposable Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ with respect to ζ, where ζ be a primitive r-th root of unity. Remark 7.7 formulates the necessary condition for a normal Gauß period α. Thus, it remains to show that $\langle q, \mathcal{K} \rangle = \mathbb{Z}_r^{\times}$ is also sufficient. Let $r_1 \cdots r_t$ be the prime power decomposition of r, and $\pi_{r_i} \colon \mathbb{Z}_r^{\times} \to \mathbb{Z}_{r_i}^{\times}$ the canonical projection, and set $n_i = \frac{\phi(r_i)}{\#\pi_{r_i}(\mathcal{K})}$ for $1 \leq i \leq t$. Proposition 9.6 states that $n_1, \ldots, n_t$ are pairwise co-prime, $n = n_1 \cdots n_t$, and $\langle q, \pi_{r_i}(\mathcal{K}) \rangle = \mathbb{Z}_{r_i}^{\times}$ for $1 \leq i \leq t$. The Gauß period α_i of type $(n_i, \pi_{r_i}(\mathcal{K}))$ over $\mathbb{F}_q$ with respect to ζ^{r/r_i} is either a prime or prime power Gauß period for $1 \leq i \leq t$. It is normal according to Theorem 8.41. By Lemma 9.5, we get α as the product of conjugates of the normal Gauß periods $\alpha_1, \ldots, \alpha_t$. With the help of Fact 7.21, we conclude inductively that α is normal in $\mathbb{F}_{q^n}$. □

We are now ready to prove Corollary 9.9.

PROOF (of Corollary 9.9). Since a normal decomposable Gauß period is in particular a normal Gauß period, it remains to show direction "(i) ⇒ (ii)". For a normal Gauß period α we have $\langle q, \mathcal{K} \rangle = \mathbb{Z}_r^{\times}$ by Remark 7.7. Then Corollary 9.11 guarantees that there is a decomposable subgroup $\mathcal{L} \subseteq \mathbb{Z}_r^{\times}$ of same order k such that $\langle q, \mathcal{L} \rangle = \mathbb{Z}_r^{\times}$. By Theorem 9.17 this condition is sufficient for a decomposable Gauß period to be normal. □

We merge Corollary 9.9 with Theorem 9.8, and apply fast polynomial multiplication to prove Result 9.1.

PROOF (of Result 9.1). Let α' be a general Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ generating a normal basis in $\mathbb{F}_{q^n}$. By Corollary 9.9 there is a normal decomposable Gauß period α of type $(n, \mathcal{L})$ in $\mathbb{F}_{q^n}$ with $\#\mathcal{L} = \#\mathcal{K}$. Thus, we can write

an element of $\mathbb{F}_{q^n}$ as a linear combination of the elements of the normal basis $\mathcal{N} = (\alpha, \ldots, \alpha^{q^{n-1}})$ over $\mathbb{F}_q$. In this case Theorem 9.8 states that we can apply fast polynomial multiplication to compute the product of two elements in $\mathbb{F}_{q^n}$. Inserting $\mathsf{M}(r_i) = O(r_i \log r_i \cdot \operatorname{loglog} r_i)$ for $1 \leq i \leq t$ due to Fact 5.8 proves the claimed bound on the number of operations in $\mathbb{F}_q$. □

9.4. A criterion for the existence of a normal Gauß period. Up to now, we have worked with the assumption that we already know a triple of integers q, n and r such that n divides $\phi(r)$ and $\mathbb{Z}_r^\times$ has a subgroup $\mathcal{K}$ of order $k = \frac{\phi(r)}{n}$. Given such a subgroup $\mathcal{K}$, it is easy to check whether the Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ is normal in $\mathbb{F}_{q^n}$; we just have to apply the criterion given in the Normal Gauß period theorem 7.5.

In practice, one starts with a finite field $\mathbb{F}_{q^n}$. Then the task is to construct a normal Gauß period in $\mathbb{F}_{q^n}$. Thus, only q and n are given. The integer r is missing. The question, whether there exists r for given q and n, was answered by Gao (2001). He generalized Fact 6.16, which was proven by Wassermann (1990).

FACT 9.18 (Gao 2001, Theorem 1.4). *Let p be a prime, n and e be positive integers, and set $q = p^e$. There exist a positive integer r and a subgroup $\mathcal{K} \subseteq \mathbb{Z}_r^\times$ of order $k = \frac{\phi(r)}{n}$ such that the Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ is normal in $\mathbb{F}_{q^n}$ if and only if q and n satisfy*

$$\gcd(e, n) = 1, \text{ and } 8 \not| n \text{ in the case } p = 2.$$

In the case of existence, one can efficiently create integers r for given q and n such that n divides $\phi(r)$ and $\gcd(r, q) = 1$. Then the order $k = \frac{\phi(r)}{n}$ of the required subgroup of $\mathbb{Z}_r^\times$ is fixed. But the search for a suitable subgroup $\mathcal{K} \subseteq \mathbb{Z}_r^\times$ of this order is still to be done. As illustrated in Example 7.4, the subgroup is not uniquely determined by its order in the general case. Moreover, the same example showed that one subgroup of given order may satisfy the criterion of Fact 6.16 but another subgroup may fail.

In this closing part of Section 9, we describe how to check efficiently whether there is a suitable subgroup $\mathcal{K}$ if q, n and r are fixed. Therefore, we do not explicitly generate and test all subgroups of $\mathbb{Z}_r^\times$ of order $k = \frac{\phi(r)}{n}$. Our approach which we present now is constructive, i.e. if there is a normal Gauß period for given integers q, r, n then a decomposable subgroup $\mathcal{L}$ of order $k = \frac{\phi(r)}{n}$ satisfying $\langle q, \mathcal{L} \rangle = \mathbb{Z}_r^\times$ can be easily derived. For some of the basic ideas see also Gao (2001), Section 5.

In the prime case, i.e. $r = p$ is a prime, the group $\mathbb{Z}_r^\times$ is cyclic. Thus, a subgroup $\mathcal{K}$ is uniquely determined by its order k. Gao *et al.* (2000), Theorem 3.1, gave a criterion in this case. Their proof holds for all cyclic groups $\mathbb{Z}_r^\times$, i.e. r is a prime power not divisible by 8.

FACT 9.19 (Gao *et al.* 2000, Theorem 3.1). *Let $r = p^e$ be a prime power not divisible by 8, and let q be an integer greater than 1 and co-prime to r. Let n be a positive divisor of $\phi(r)$, and $\mathcal{K}$ the uniquely determined subgroup of $\mathbb{Z}_r^\times$ of order $k = \frac{\phi(r)}{n}$. Then $\langle q, \mathcal{K}\rangle = \mathbb{Z}_r^\times$ if and only if $\gcd(\frac{\phi(r)}{N}, n) = 1$, where $N = \mathrm{ord}_r(q)$ is the order of q in $\mathbb{Z}_r^\times$.*

PROOF. Let ξ be a generator of the cyclic group $\mathbb{Z}_r^\times$. Then the subgroups $\langle q\rangle$ and $\mathcal{K}$ are both cyclic with generators $\xi^{\phi(r)/N}$ and $\xi^{\phi(r)/k}$, respectively. By assumption, we have $\frac{\phi(r)}{k} = n$. The subgroup $\langle q, \mathcal{K}\rangle$ of $\mathbb{Z}_r^\times$ contains all elements of type ξ^f with $f = a \cdot \frac{\phi(r)}{N} + b \cdot n$ for $a, b \in \mathbb{Z}$. As a consequence of the Extended Euclidean Algorithm, we have $f = 1$ if and only if $\gcd(\frac{\phi(r)}{N}, n) = 1$, and the claim follows. □

For the non-cyclic group $\mathbb{Z}_{2^e}^\times$ with $e \in \mathbb{N}_{\geq 3}$ this criterion is no longer true.

EXAMPLE 9.20. For $r = 8$ and $\mathcal{K} = \{1, 7\}$, we have $\langle 3, \mathcal{K}\rangle = \{1, 3, 5, 7\} = \mathbb{Z}_8^\times$, i.e. $\frac{\phi(8)}{\#\mathcal{K}} = \frac{4}{2} = 2$. Furthermore, $N = \mathrm{ord}_8(3) = 2$, i.e. $\frac{\phi(8)}{N} = 2$, and $\gcd\left(\frac{\phi(8)}{N}, \frac{\phi(8)}{\#\mathcal{K}}\right) = \gcd(2, 2) = 2 \neq 1$. ◊

For $n = 1$ and $k = \#\mathbb{Z}_{2^e}^\times$, we can always choose the trivial subgroup $\mathcal{K} = \mathbb{Z}_{2^e}^\times$ to get $\langle q, \mathcal{K}\rangle = \mathbb{Z}_{2^e}^\times$. For $n \geq 2$ we recall that $\mathbb{Z}_{2^e}^\times$ is the direct product of the two cyclic groups $\pm 1 = \langle -1 \bmod 2^e\rangle$ and $\mathcal{Z}_{2^e} = \langle 5 \bmod 2^e\rangle$ as stated in Fact 9.12.

We start with the assumption that the subgroup generated by q has maximal possible order, i.e. $N = \mathrm{ord}_{2^e}(q)$.

PROPOSITION 9.21. *Let r be a power of 2 greater than 8, and let $q \geq 3$ be a positive integer co-prime to r. If $N = \mathrm{ord}_r(q) = 2^{e-2}$ and $2 \leq n \leq 2^{e-2}$ is a divisor of N then $\mathcal{K} = \pm 1 \cdot \langle 5^n \bmod 2^e\rangle$ is a subgroup of $\mathbb{Z}_r^\times$ of order $k = \frac{\phi(r)}{n}$ such that $\langle q, \mathcal{K}\rangle = \mathbb{Z}_r^\times$.*

PROOF. For $r = 2^e$ and $e \geq 4$, the subgroup $\mathcal{K}$ of $\mathbb{Z}_r^\times$ has order $2 \cdot \frac{2^{e-2}}{n} = \frac{2^{e-1}}{n} = \frac{\phi(r)}{n}$. We have $\#\langle q\rangle = N = 2^{e-2}$, by assumption. Thus, $\langle q\rangle / \pm 1 = \mathcal{Z}_{2^e}$ because q generates a cyclic subgroup. By construction, $-1 \in \mathcal{K}$, hence $\langle q\rangle \cup (-1) \cdot \langle q\rangle$ is a subset of $\langle q, \mathcal{K}\rangle$ of order $2 \cdot 2^{e-1}$. We conclude that $\#\langle q, \mathcal{K}\rangle = \phi(r)$, i.e. $\langle q, \mathcal{K}\rangle = \mathbb{Z}_r^\times$ as claimed. □

In fact, this proposition covers all cases $n \geq 4$.

REMARK 9.22. *Let e be an integer greater than 4 and set $r = 2^e$. Let q be an odd prime power and $\mathcal{K}$ be a subgroup of order k of $\mathbb{Z}_{2^e}^\times$ such that $\langle q, \mathcal{K}\rangle = \mathbb{Z}_{2^e}^\times$, and $n = \frac{\phi(2^e)}{k}$. If $n \geq 4$ then $\langle q\rangle$ has maximal order $N = \mathrm{ord}_{2^e}(q) = 2^{e-2}$.*

PROOF. Since n divides N, we have $N \geq 4$. Furthermore, the subgroup $\mathcal{K}$ has order $\#\mathcal{K} = \frac{\phi(2^e)}{n} \leq \frac{2^{e-1}}{4} = 2^{e-3} \geq 2$. Let $\bar{\cdot}\colon \mathbb{Z}_{2^e}^\times \to \mathcal{Z}_{2^e}$ be the canonical projection with $a = \bar{a} \cdot \pm 1$. Then $\langle \bar{q} \rangle$ is a cyclic subgroup of $\mathcal{Z}_{2^e}$ of order $N \geq 4$. The projection is an epimorphism. Hence, $\langle \bar{q}, \bar{\mathcal{K}} \rangle = \mathcal{Z}_{2^e}$. But $n' = \#\mathcal{Z}_{2^e}/\#\bar{\mathcal{K}} \geq \frac{2^{e-2}}{2^{e-3}} = 2$ is divisible by 2, and by Remark 9.13 the subgroup $\langle \bar{q} \rangle$ contains a subgroup of maximal order 2^{e-2}, since $\mathcal{Z}_{2^e}$ is cyclic. We conclude that $\langle \bar{q} \rangle = \mathcal{Z}_{2^e}$, i.e. $N = \operatorname{ord}_{2^e}(q) \geq \#\mathcal{Z}_{2^e} = 2^{e-2}$. But a cyclic subgroup of $\mathbb{Z}_{2^e}^\times$ has order at most 2^{e-2} and thus $N = 2^{e-2}$. □

For $e = 3$, we have always $N = 2$, and there is a subgroup $\mathcal{K} \subseteq \mathbb{Z}_8^\times$ of order 2 with $\langle q, \mathcal{K} \rangle = \mathbb{Z}_8^\times$; for given $q \geq 3$ we can choose $\mathcal{K} = \langle a \rangle$ with $a \in \mathbb{Z}_8^\times \setminus \{1, q \bmod 8\}$.

The only case left is $n = 2$ and $2 \leq N < 2^{e-2}$ for $e \geq 4$. Here two different cases of q are important. Since we have q an odd prime power, either $q \equiv 1 \bmod 4$ or $q \equiv 3 \bmod 4$. These two cases have different projections of $\langle q \rangle$ onto ± 1. We regard the canonical projection $\pi\colon \mathbb{Z}_{2^e}^\times \to \mathbb{Z}_4^\times$. Then $\ker \pi = \mathcal{Z}_{2^e}$, and we have a bijection between $\pm 1 = \mathbb{Z}_{2^e}^\times / \ker \pi = \mathbb{Z}_{2^e}^\times / \mathcal{Z}_{2^e}$ and $\mathbb{Z}_4^\times$ applying the fundamental theorem on groups. Thus, $\langle q \rangle / \mathcal{Z}_{2^e} = \pm 1$ if $q \equiv 3 \bmod 4$ and it is equal $\{1 \bmod 2^e\}$ if $q \equiv 1 \bmod 4$.

PROPOSITION 9.23. *Let e be a positive integer greater or equal 4 and set $r = 2^e$. Let q be an integer greater than 2 and co-prime to r. Let $n = 2 \leq N = \operatorname{ord}_r(q) < 2^{e-2}$. Then there is a subgroup $\mathcal{K} \subseteq \mathbb{Z}_{2^e}^\times$ of order k such that $\langle q, \mathcal{K} \rangle = \mathbb{Z}_r^\times$ if and only if $q \equiv 3 \bmod 4$.*

PROOF. For $q \equiv 3 \bmod 4$, we have $\langle q \rangle / \mathcal{Z}_{2^e} = \pm 1$. Since $n = 2 = \# \pm 1$ and $\pm 1 \subseteq \langle q \rangle$ by assumption, we have $\langle q^{N/n} \rangle = \pm 1$. Choosing the subgroup $\mathcal{K} = \mathcal{Z}_{2^e}$ of order $k = 2^{e-2}$ gives $\langle q, \mathcal{K} \rangle = \mathbb{Z}_{2^e}^\times$.

For $q \equiv 1 \bmod 4$, we have $\langle q^{N/n} \rangle = \langle 5^{2^{e-3}} \bmod 2^e \rangle = \{5^{2^{e-3}}, 1\} \subset \mathbb{Z}_{2^e}^\times$. Since $e \geq 4$, there are three subgroups of $\mathbb{Z}_{2^e}^\times$ of order $k = 2^{e-2} \geq 4$ in this case: $\mathcal{K}_1 = \langle 5 \bmod 2^e \rangle$, $\mathcal{K}_2 = \langle -5 \bmod 2^e \rangle$, and $\mathcal{K}_3 = \pm 1 \cdot \langle 5^2 \bmod 2^e \rangle$. For $e \geq 4$, we have $2^{e-3} \geq 2$ and $5^{2^{e-3}} = (-5)^{2^{e-3}} = (5^2)^{2^{e-4}} \bmod 2^e$ is an element of all three subgroups. Hence, $\langle q, \mathcal{K}_i \rangle = \mathcal{K}_i \neq \mathbb{Z}_{2^e}^\times$ for $1 \leq i \leq 3$. Thus, there is no suitable subgroup in the case $q \equiv 1 \bmod 4$. □

We collect the findings above to get the following criteria on the existence of a suitable subgroup $\mathcal{K}$ in $\mathbb{Z}_{2^e}^\times$.

LEMMA 9.24. *Let r be a power of two divisible by 8. Let $q > 1$ be an odd integer, and n be a divisor of $N = \operatorname{ord}_r(q)$. Set $k = \frac{\phi(r)}{n}$. Then the following are equivalent:*

(i) There is a subgroup $\mathcal{K} \subseteq \mathbb{Z}_r^\times$ of order k with $\langle q, \mathcal{K} \rangle = \mathbb{Z}_r^\times$.

(ii) One of the following criteria holds:

- $n = 1$, *or*
- $n = 2$ *and* $q \equiv 3 \bmod 4$, *or*
- $N = 2^r/4$.

PROOF. We write $r = 2^e$ with $e \geq 2$. If one of the criteria in (ii) is satisfied then either $n = 1$ and $\mathcal{K} = \mathbb{Z}_{2^e}^\times$, or Proposition 9.21 or Proposition 9.23, respectively, guarantee the existence of a subgroup $\mathcal{K}$ of order k with $\langle q, \mathcal{K}\rangle = \mathbb{Z}_{2^e}^\times$ for $e \geq 4$. There are two more cases to consider. For $e = 3$ and $n = 2$ we have $N = \mathrm{ord}_8(q) = 2$. Then we can choose $\mathcal{K} = \{1, 3\}$ if $q \equiv 1 \bmod 4$ and $\mathcal{K} = \{1, 5\}$ if $q \equiv 3 \bmod 4$. Thus, it remains to prove that in the case $n = 2$ and $q \equiv 1 \bmod 4$ and $N < 2^{e-2}$ there is no suitable subgroup. We have $\langle q\rangle / \pm 1 \subset \mathcal{Z}_{2^e}$, and thus $\langle q\rangle \subseteq \langle 5^2 \bmod 2^e\rangle$. But $5^2 \bmod 2^e$ is an element in all three subgroups of order $k = 2^{e-2}$ of $\mathbb{Z}_{2^e}^\times$; we have $5^2 \in \langle 5 \bmod 2^e\rangle$ and $5^2 = (-5)^2 \in \langle -5 \bmod 2^e\rangle$ and $1 \cdot 5^2 \in \pm 1 \cdot \langle 5^2 \bmod 2^e\rangle$. Since we have discussed all possible cases, equivalence holds. □

Merging these criteria with Theorem 9.17 proves an alternative criterion. Applying this criterion, we can get rid of a preliminary fixing of a subgroup $\mathcal{K}$.

THEOREM 9.25. *Let q be a prime power and r and n be positive integers such that $\gcd(r, q) = 1$ and n divides $\phi(r)$. Let $k = \frac{\phi(r)}{n}$ and $r_1 \cdots r_t$ be the prime power decomposition of r. Then the following are equivalent:*

(i) There is a subgroup $\mathcal{K}$ of $\mathbb{Z}_r^\times$ of order k such that the Gauß period α of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ is normal.

(ii) There are pairwise co-prime positive integers $n_1, \ldots, n_t$ such that $n = n_1 \cdots n_t$, and

- $\gcd(\frac{\phi(r_i)}{N_i}, n_i) = 1$ *if r_i is not divisible by 8, and*
- *n_i divides N_i and either $n_i = 1$, or $n_i = 2$ and $q \equiv 3 \bmod 4$, or $N_i = 2^{e-2}$ if 8 divides r_i*

where $N_i = \mathrm{ord}_{r_i}(q)$ for $1 \leq i \leq t$.

PROOF. "(i)⇒(ii)" By Theorem 9.17 there is a decomposable Gauß period of type $(n, \mathcal{L})$ over $\mathbb{F}_q$ with $\langle q, \mathcal{L}\rangle = \mathbb{Z}_r^\times$. By Proposition 9.6 the $n_i = \frac{\phi(r_i)}{\#\pi_{r_i}(\mathcal{L})}$ for $1 \leq i \leq t$ are pairwise co-prime and $n_1 \cdots n_t = n$. Furthermore, $\langle q, \pi_{r_i}(\mathcal{L})\rangle = \mathbb{Z}_{r_i}^\times$ and the criteria follows immediately with Fact 9.19 and Lemma 9.24.

"(ii)⇒(i)" By Fact 9.19 and Lemma 9.24, respectively, there is a subgroup $\mathcal{L}_i$ of order $k_i = \frac{\phi(r_i)}{n_i}$ such that $\langle q, \mathcal{L}_i\rangle = \mathbb{Z}_{r_i}^\times$ for all $1 \leq i \leq t$. Obviously,

Existence of normal bases generated by a general Gauß period with given parameter $k \in \mathbb{N}_{\geq 1}$								
$k \quad \backslash \quad q$	2	3	5	7	11	13	17	19
$k = 1$	4.90	4.91	5.10	4.80	4.60	4.76	4.65	4.83
$k \leq 2$	26.78	29.42	25.79	27.12	25.60	24.76	26.27	25.26
$k \leq \log_2 n$	76.52	87.83	87.48	87.71	87.18	86.60	86.80	86.71
$k \leq \sqrt{n}$	87.30	99.76	99.75	99.71	99.74	99.71	99.62	99.67
$k < \infty$	87.51	100.00	100.00	100.00	100.00	100.00	100.00	100.00
theo.	87.51	100.00	100.00	100.00	100.00	100.00	100.00	100.00

Table 9.4: Percentage of field extensions $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ with $n \leq 10000$ for which there is a normal basis given by a general Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$. Each column is related to a prime field $\mathbb{F}_q$, with q given in the first row. The rows show the distribution if the value for $k = \mathcal{K} \in \mathbb{N}_{\geq 1}$ is also limited. We limited our experiments for r with $\phi(r) = nk$ by $2 \leq r < 1000000$. The last row labeled *theo.* is due to Fact 9.18.

$\mathcal{L} = \mathcal{L}_1 \times \cdots \times \mathcal{L}_t$ meets the assumptions of Proposition 9.6. By Theorem 9.17, the criterion $\langle q, \mathcal{L} \rangle = \mathbb{Z}_r^\times$ is sufficient for the Gauß period of type $(n, \mathcal{L})$ over $\mathbb{F}_q$ to be normal.

□

We close this section by computing the distribution of normal general Gauß periods for prime fields $\mathbb{F}_q$ with $2 \leq q < 20$ and field extensions $2 \leq n < 10000$. The values are listed in Table 9.4. We have applied the criterion given in Theorem 9.25 to compute the percentage of the extension fields over $\mathbb{F}_q$ for which there is a normal Gauß period.

10. Scalable parallel exponentiation

In Section 4.1 we have presented an algorithm for the parallel computation of an e-th power that is related to the quotient $c = \frac{c_Q}{c_A} \geq 0$. The algorithm works with a fixed number of processors that depends only on the inputs $\lambda_q(e)$ and c. With the help of this algorithmic approach, we have discussed the possible speed-up for exponentiation with respect to different basis representations of $\mathbb{F}_{q^n}$. Our previous results may support principal design decisions when implementing e.g. cryptography over finite fields.

In this section we describe our ideas and some details when implementing parallel exponentiation algorithms on real parallel machines. We develop a *scalable* parallel algorithm for exponentiation. A similar approach was presented by Stinson (1990) for exponentiation in $\mathbb{F}_{2^n}$ given by a normal basis representation. He called such algorithms *adaptive*[62]. Restrictions on the number of processors were also discussed by von zur Gathen (1992). Such restrictions pay attention to the fact that real parallel machines consist only of a fixed number of processors. In particular, the assumption that the number of processors is scalable due to the input size is often too generous. On the other hand, automatic distribution of a large number of virtual processors on the small number of given processors might cause poor results. If we would run the two virtual processors needed for Algorithm 4.3 on a single-processor machine, then we run in fact the binary addition chain in sequential. But this is slower than Brauer's algorithm; the latter one gives a speed-up of about 1.3 on average compared to the binary addition chain, see Figure 10.1. Thus, we add the number of processors as a new parameter to get the following:

RESULT 10.1. *Let e be a positive integer, and c_Q and c_A be the number of operations in $\mathbb{F}_q$ to evaluate a q-th power, and to multiply two elements in $\mathbb{F}_{q^n}$, respectively. Let P with $1 \leq P \leq n$ be the number of processors. Then the power e of an element in $\mathbb{F}_{q^n}$ can computed in depth at most*

$$c_A \cdot \left(2 \cdot \lceil \log_2 P \rceil + 4 + \frac{1}{P} \cdot \frac{n}{\log_q n} \cdot (1 + o(1))\right)$$

operations in $\mathbb{F}_q$ using at most P processors if $c_Q = 0$. If $c_Q > 0$ then the depth is bounded by

$$c_Q \cdot (n-1) + c_A \cdot \left(2 \cdot \lceil \log_2 P \rceil + 3 + \frac{c}{(1+c)^P - 1} \cdot \frac{n}{\log_q n} \cdot (1 + o(1))\right)$$

operations in $\mathbb{F}_q$ if $P \geq \frac{n}{3 \log_q n}$. Otherwise the depth is at most

$$\begin{aligned} & c_Q \cdot (n-1) + c_A \cdot \left(2 \cdot \lceil \log_2 P \rceil + 2 + \tfrac{c}{(1+c)^P - 1} \cdot \tfrac{n}{\log_q n} \cdot (1 + o(1))\right) \\ + \; & \tfrac{1}{P} \cdot \tfrac{n}{\log_q^2 n} \cdot (c_A \cdot (q-1) + 2c_Q) + c_Q \cdot O(\log^2 n) \end{aligned}$$

[62] "These algorithms are adaptive, in that we can vary the number of processors to suit our needs.", Stinson (1990), p. 716.

operations in $\mathbb{F}_q$ using at most P processors.

Parts of this section were already published in Nöcker (2000). The algorithm used to prove Result 10.1 is an adaption of Brauer addition chains to parallel exponentiation. We discuss the main stage of the algorithm (Section 10.2) and the precomputation (Section 10.3) separately. As a preparation, we revisit the model of weighted q-addition chains (Section 10.1) and complete it with respect to the two most important restrictions of real machines: a *limited number of processors* and *communication delay*. In the last part, we combine the different basis representations of $\mathbb{F}_{q^n}$, that have been analyzed in the previous sections, with Result 10.1. This comparison completes our discussion on parallel exponentiation.

10.1. A refined model. Our algorithm in Section 4.1 works with a fixed number of processors and neglects communication cost. On real parallel computers the number of processors is limited. This can influence algorithmic design even for moderate input size of $\lambda_q(e)$.

For example Algorithm 4.14 computes a 2-addition chain with weight $(0, 1)$ for an exponent of size 2048 bits on at most 1024 processors. The largest modern parallel system available at the University of Paderborn has 96 computing units with a total of 192 processors available. Thus, we have to adapt the original algorithm with respect to the *number of processors*.

But even if there are enough processors available—as for Algorithm 4.3 if $P \geq 2$—a redesign with respect to real-existing machines may improve the times. This is because most parallel computers, and also computer networks, are a collection of basic units which consist of local memory and a few processors (typically one or two). Locally calculated and stored intermediate results are made available to other processors via the network. This principal design of parallel systems causes *communication delay* when accessing to non-local storage. Algorithm 4.3 has optimal depth if we neglect communication cost. It tends to exchange a lot of small pieces of information between the two involved processors. This extensive communication can slow down the whole parallel computation. Figure 10.1 illustrates that a substitution of communication by computation may be better for small input size even if the original algorithm is optimal in the PRAM model that neglects communication cost. Figure 10.1 illustrates that it may yield advantages to substitute extensive communication of small pieces of data between processors by redundant computation of intermediate results. We extend our model of parallel computation with respect to this observation.

Since we want to keep our refined model handily, we abstract from technical details. In particular, it should not be burden by network layout and such things. Furthermore, communication should not affect the analysis of those parts of the algorithm where each processor works only with locally stored data.

A model that meets our requirements was invented by Valiant (1990). We de-

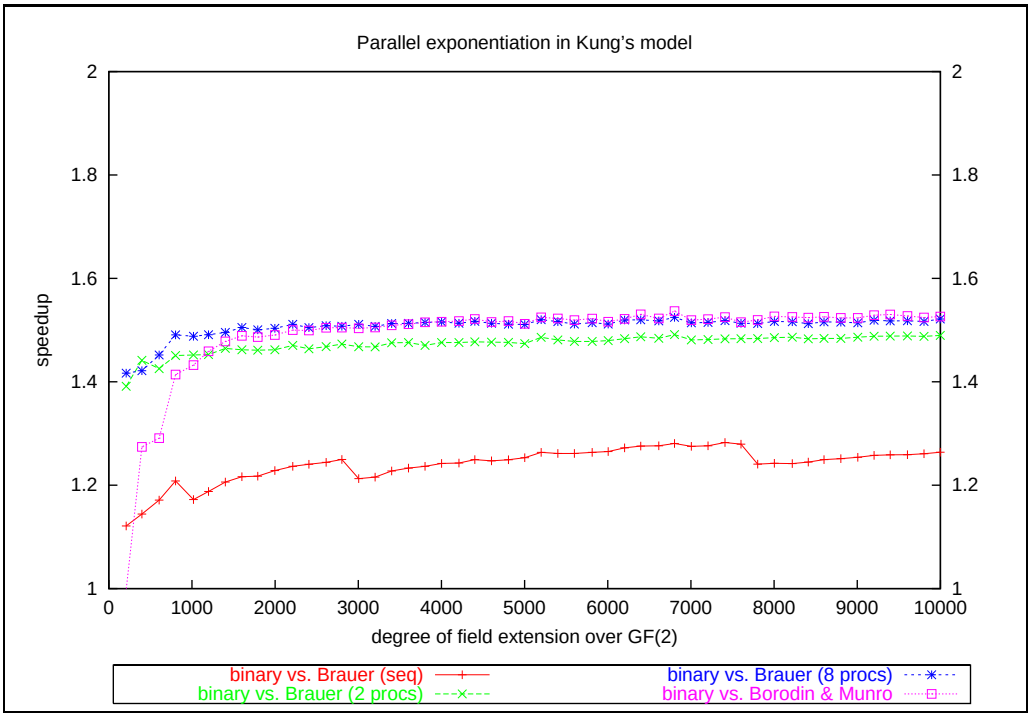


Figure 10.1: Parallel exponentiation in the model where the only operation is a multiplication. All speed-ups are relative to the binary addition chain, i.e. the repeated squaring algorithm. This chain is compared to Brauer addition chains. Beside the sequential algorithm (+), we tested a parallel version of Brauer's algorithm with 2 ($\times$) and 8 ($*$) processors, respectively. Furthermore, we compared repeated squaring to Algorithm 4.3 ($\square$). The latter one uses 2 processors and is optimal for original additin chains if one neglects communication cost. The experiment shows that this assumption is too optimistically for field extensions with small degree n.

scribe his *bulk synchronous parallel* (or BSP for short) model on the background of exponentiation. The model is characterized by three parameters[63] which determine the *parallel environment*:

1. The *number of processors*[64] $P \in \mathbb{N}_{\geq 1}$: each processor has local memory. All processors are connected to non-local memory by the network.

2. The *latency* and startup cost for routing is described by the parameter $\ell \in \mathbb{Q}_{\geq 0}$. Here it is given as the number of non-q-steps that can be performed in the same time as synchronizing the network.

[63] Valiant describes the three attributes and the resulting analysation parameters in two steps, see Valiant (1990), p. 105 first column and last column. We mix both parts.

[64] Valiant (1990), p. 105, calls them *components*.

3. Finally, $g \in \mathbb{Q}_{\geq 0}$ contains the *amount of communication*. Originally, the parameter $g \in \mathbb{Q}_{\geq 0}$ describes the basic throughput of the router. It can be described by the ratio of basic computational operations, to the total throughput of the router in terms of words of information delivered.[65] Our basic computational operation is the calculation of a non-q-step. Our basic unit of delivered information is one element of the addition chain already computed. Hence in what follows, g is the number of non-q-steps multiplied with its weight c_A that can be performed in the same time as one element A^a of the underlying (cyclic) multiplicative group $\langle A \rangle \subseteq \mathcal{G}$—represented by its exponent $a \in \mathbb{N}_{\geq 0}$—can be copied between different processors through the network.

Valiant (1990) subdivided the total parallel computation into *supersteps*. Each superstep consists of a local (sequential) computation on each single processor and a subsequent data exchange. At the beginning, all processors know the exponentiation problem: an element $A \in \mathcal{G}$, represented by the empty addition chain γ with $\mathcal{S}(\gamma) = \{1\}$, and the exponent $e \in \mathbb{N}_{\geq 1}$. Intermediate elements of the q-addition chain that are not originally computed by a processor itself are available to this processor earliest after the superstep they are computed at. A superstep $1 \leq s \leq S$ has communication cost at most $\ell + g \cdot t_s$ where t_s is the maximum number of received or sent exponents over all processors. The *total communication cost* C of an addition chain γ is the sum over all supersteps:

$$C_{\ell,g}(\gamma) = \sum_{1 \leq s \leq S} (\ell + g \cdot t_s) = S \cdot \ell + g \cdot t \text{ with } t = \sum_{1 \leq s \leq S} t_s.$$

Addition chains and the BSP-model. We use the BSP-model to take a limited number of processors and communication delay into consideration. We have to refine our model of addition chains once more, because we are forced to do book-keeping: Which processors have computed an element that appears in the semantics $\mathcal{S}(\gamma) = \{a_0, \ldots, a_l\}$ of a q-addition chain γ? Thus, we connect an element $(j(i), k(i))$ of an addition chain γ to a processor $p(i) \in \mathcal{P} = \{0, \ldots, P-1\}$. Since a processor should store all its elements, we assume $a_i \neq a_{i'}$ if $p(i) = p(i')$ for $1 \leq i \leq l$. We still will describe a q-addition chain by its semantics and the set of rules. But we use a slightly modified notation. Now, a node is labeled by the pair $(a_i, p(i)) \in \mathcal{S} \times \mathcal{P}$ for $1 \leq i \leq l$. We suppose the nodes $(1,0), \ldots, (1, P-1)$ to be given. The set of rules $\mathcal{R}$ is now a subset of $(\mathcal{S} \times \mathcal{P}) \times ((\mathcal{S} \times \mathcal{P}) \cup \{-q\})$. For a q-step with $k(i) = -q$ we include $((a_{j(i)}, p(j(i))), -q)$ in $\mathcal{R}$; a non-q-step with $k(i) \neq -q$ is denoted by the rule $((a_{j(i)}, p(j(i))), (a_{k(i)}, p(k(i))))$.

A *copy step* is given if $p(i)$ and $p(j(i))$ or $p(k(i))$ for $k(i) \neq -q$ differ. If both $p(j(i))$ and $p(k(i))$ are different then there are two copy steps. Since there

[65] "This g can be regarded as the ratio of the number of local computational operations performed per second by all the processors, to the total number of data words delivered per second by the router.", Valiant (1990), p. 106.

is local storage, a node $(a_{j(i)}, p(j(i)))$ is copied to processor $p(i)$ at most once. In the BSP-model, there has to be a synchronization between the calculation of an element $a_{i'}$ by processor $p(i')$ and its usage by processor $p(i) \neq p(i')$. This synchronization is realized by dividing the addition chain into supersteps. The earliest superstep in which a node $(a_i, p(i))$ for $1 \leq i \leq l$ can be used is given as follows: The node $(1, p)$ with $0 \leq p < P$ is computed in superstep $s(1, p) = 1$. For a node $(a_i, p(i))$ with $1 \leq i \leq l$, we set

$$s(a_i, p(i)) \quad = \quad \begin{cases} s(a_{j(i)}, p(j(i))) + |\mathrm{sign}(p(i) - p(j(i)))| & \text{if } k(i) = -q, \\ \max\{s(a_{j(i)}, p(j(i))) + |\mathrm{sign}(p(i) - p(j(i)))|, & \\ \qquad s(a_{k(i)}, p(k(i))) + |\mathrm{sign}(p(i) - p(k(i)))|\} & \text{else.} \end{cases}$$

We say that a q-addition chain γ can be computed in (at least) $S \in \mathbb{N}_{\geq 1}$ supersteps if

$$S \quad \geq \quad \max\{s(a_i, p(i)) \colon a_i \in \mathcal{S}(\gamma)\}.$$

The total cost for a parallel weighted q-addition chain γ computing e in parallel are the sum of the weighted depth $\delta_{q,(c_Q,c_A)}(\gamma) = c_Q \cdot Q + c_A \cdot A$ and the communication cost $C_{\ell,g}(\gamma) = \ell \cdot S + g \cdot t$. Subsequently, we mainly focus on the depth $\delta_{q,(c_Q,c_A)}(\gamma)$ of an addition chain γ. We also analyze the communication cost with respect to the BSP-model. We bear in mind two basic strategies as contribution to the communication costs.

1. We try to *avoid communication*, and we try to bundle as much data as possible before we initiate a new superstep.

2. We try to *avoid idle times*. It may be better to compute an intermediate result locally if we can substitute communication by idle times.

10.2. The main stage. Now, we develop an algorithm for parallel exponentiation that is scalable with respect to the number of processors. We follow Brauer's approach and divide the algorithm into a precomputation stage and a main part; this is the same strategy as in Section 4.1. The connecting link between both parts is the q^m-ary representation of the exponent. The tuning parameter $m \in \mathbb{N}_{\geq 1}$ allows a tradeoff between both stages. We start with the description of the main stage. The precomputation is supposed as an input here, it will be discussed at length in the subsequent Section 10.3.

Basic assumptions. For the main stage, we fix $q^m \in \mathbb{N}_{\geq 2}$. We focus on the q^m-ary representation $(e)_{q^m} = (e_{\lambda-1}, \ldots, e_0)$ of the exponent $e \in \mathbb{N}_{\geq 1}$. Furthermore by precomputation, all processors $0 \leq p < P$ have stored the elements $\{1, \ldots, q^m - 1\}$ locally.

As claimed above, the number of processors P is supposed to be limited. Since the digits of the q^m-ary representation of e are our atomic units, we restrict the

number of processors P to $\lambda_{q^m}(e)$. To simplify the notation, we write q instead of q^m in this subsection but bear in mind the tuning parameter m.

We introduce some further notation: Let

$$0 = i(0) < i(1) < \cdots < i(P-1) < i(P) = \lambda_q(e) \tag{10.2}$$

be an increasing sequence of integers, and we set

$$E_p = \sum_{i(p) \le j < i(p+1)} e_j q^j,$$

which is containing a part of the q-ary representation of e for all $0 \le p < P$. We call the sequence (10.2) a *partition of the (q-ary representation of the) exponent* e if $e = \sum_{0 \le p < P} E_p$. For $0 \le p < P$ each part E_p includes

$$\Lambda_p = i(p+1) - i(p)$$

digits of $(e)_q$, and a sedimentary part of $i(p)$ zeros. Informally spoken, this sedimentary part marks the number of q-steps that are necessary to shift the Λ_p leading digits of E_p to the right position in $(e)_q$.

EXAMPLE 10.3. The exponent $(53)_2 = (110101)$ has binary length $\lambda_2(53) = 6$.

(i) The *uniform partition* for 3 processors is given by

$$i(0) = 0 < i(1) = 2 < i(2) = 4 < i(3) = 6, \text{ i.e. } (11|01|01)$$

and $\Lambda_0 = \Lambda_1 = \Lambda_2 = 2$. The intermediate results for 53 are $E_0 = 1 \cdot q^0 + 0 \cdot q^1$, $E_1 = 1 \cdot q^2 + 0 \cdot q^3$ and $E_2 = 1 \cdot q^4 + 1 \cdot q^5$.

(ii) Another partition is

$$i(0) = 0 < i(1) = 3 < i(2) = 5 < i(3) = 6, \text{ i.e. } (1|10|101).$$

In this case we have $\Lambda_0 = 3$, $\Lambda_1 = 2$ and $\Lambda_2 = 1$, and the intermediate results are $E_0 = 1 + q^2$, $E_1 = q^4$ and $E_2 = q^5$. ◊

The algorithm. As for Algorithm 4.12, we separate the local computation and the collection. Thus, the processor labeled p computes E_p. In a second step, all intermediate results are collected. Processor 0 is assumed to store the final result $e = \sum_{0 \le p < P} E_p$. As stated above, the algorithm is basically a parallel version of Algorithm 3.12.

The common way to collect results in parallel is by a (complete) t-tree. A t-tree is a tree where all inner nodes have at most $t \in \mathbb{N}_{\ge 2}$ children. Our main stage is the following algorithm.

ALGORITHM 10.4. Parallel exponentiation with limits: main stage.

Input: A scalar $q \in \mathbb{N}_{\geq 2}$, an exponent $e \in \mathbb{N}_{\geq 1}$ with $(e)_q = (e_{\lambda-1}, \ldots, e_0)$, the number $P \in \mathbb{N}_{\geq 1}$ processors, and an addition chain γ' with $\{(e_i, p)\colon (e)_q = (e_{\lambda-1}, \ldots, e_0), 0 \leq p < P\} \subseteq \mathcal{S}(\gamma')$.

Output: A parallel q-addition chain γ computing e using P processors.

1. Set $\gamma = \gamma'$.
2. Determine a partition $0 = i(0) < \cdots < i(P) = \lambda_q(e)$ for $(e)_q$.
3. For all processors $0 \leq p < P$ in parallel do 4–10
   Compute a q-addition chain for $E_p = \sum_{i(p)\leq j<i(p+1)} e_j q^j$ on processor p.
4. If $\sum_{i(p)\leq j<i(p+1)} e_j q^j \neq 0$ then
5. set $i = \max\{i(p) \leq j < i(p+1)\colon e_j \neq 0\}$ and $E_p = e_i$.
6. For j from $i-1$ down to $i(p)$ do 7–8
7. Set $E_p \leftarrow E_p \cdot q$. Add node (E_p, p) and rule $((E_p/q, p), -q)$ to γ.
8. Set $E_p \leftarrow E_p + e_j$. If $(E_p, p) \notin \mathcal{S}(\gamma) \times \{p\}$ then add node (E_p, p) and rule $((E_p - e_j, p), (e_j, p))$ to γ.
9. For $0 \leq j < i(p)$ do set $E_p \leftarrow E_p \cdot q$ and add node (E_p, p) and rule $((E_p/q, p), -q)$ to γ.
10. Else set $E_p = 0$.
    Perform a (complete) t-tree to get e.
11. For $0 \leq l < \lceil \log_t P \rceil$ do 12–14
12. If t^{l+1} divides p then
13. For $1 \leq u < t$ do
14. If $p + ut^l < P$ and $E_{p+ut^l} > 0$ then set $E_p \leftarrow E_p + E_{p+ut^l}$ and add node (E_p, p) and rule $((E_p - E_{p+ut^l}, p), (E_{p+ut^l}, p + ut^l))$ to γ.
15. Return γ.

EXAMPLE 10.3 CONTINUED. We apply the algorithm to compute a 2-addition chain with weight $(1, 2)$ for 53 using 3 processors. We use a binary tree in steps 11–14 and compare the two partitions computed above for step 2. The resulting addition chains are given in Figure 10.2. ◊

LEMMA 10.5. *The algorithm works as specified.*

(i) It generates a q-addition chain γ with weight (c_Q, c_A) computing e in depth at most

$$\begin{aligned}\delta_{q,(c_Q,c_A)}(\gamma) \quad &\leq \quad \max_{0\leq p<P} \{c_A \cdot (i(p+1) - i(p) - 1) + c_Q \cdot (i(p+1) - 1)\} \\ &\quad + c_A \cdot \lceil \log_t P \rceil \cdot (t-1).\end{aligned}$$

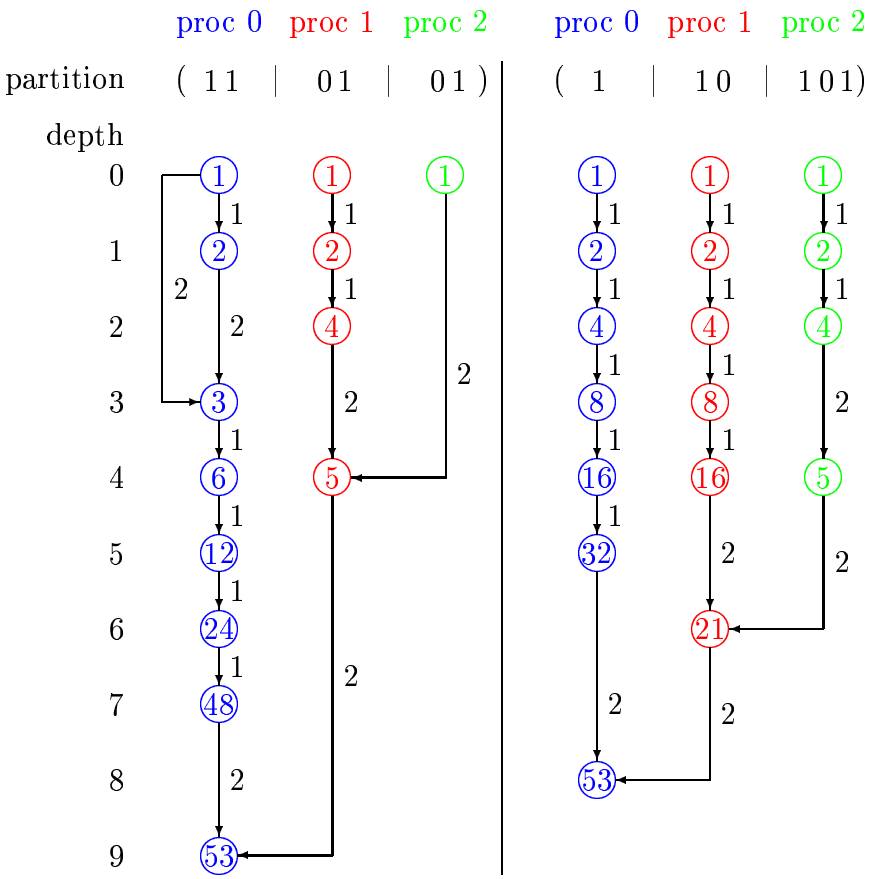


Figure 10.2: Parallel 2-addition chains with weight $(1, 2)$ for 53. The partitions are described in Example 10.3. The left graph is with respect to the uniform partition, the right one takes the weight into account.

(ii) In the BSP-model this can be implemented with $S = \lceil \log_t P \rceil + 1$ *supersteps and communication cost*

$$C_{\ell,g}(\gamma) \leq \lceil \log_t P \rceil \cdot (\ell + g \cdot (t-1)) .$$

Proof. We prove correctness in two steps. For the loop in steps 6–8, we have $E_p = \sum_{j \leq h < i(p+1)} e_h q^{h-j}$ after lap j for processor $0 \leq p < P$. Thus, after lap $i(p)$ we have $E_p = \sum_{i(p) \leq h < i(p+1)} e_h q^{h-i(p)}$. In step 9, the algorithm adds $i(p)$ many q-steps, i.e. after the local computation, we have $E_p = q^{i(p)} \cdot \sum_{i(p) \leq h < i(p+1)} e_h q^{h-i(p)} = \sum_{i(p) \leq h < i(p+1)} e_h q^h$ as claimed.

For the collection part in steps 11–14, we have the following invariant which can be shown by induction on i: If $p = ut^l$ for an $u \in \mathbb{N}_{\geq 0}$ and $0 \leq l \leq \lceil \log_t P \rceil$ then $E_p = \sum_{i(p) \leq j < \min\{i(p+ut^{l+1}-1), i(P)\}} e_j q^j$ after lap l. Thus, after the last lap processor 0 has computed the node $E_0 = \sum_{i(0) \leq j < \min\{i(t^{\lceil \log_t P \rceil}), i(P)\}} e_j q^j =$

$\sum_{0\le j<i(P)} e_j q^j = \sum_{0\le j<\lambda_q(e)} e_j q^j = e$.

We count all q-steps and non-q-steps in the loop (steps 6–9). The loop can be computed in depth at most

$$\begin{aligned} & (c_Q + c_A) \cdot (i(p+1) - i(p) - 1) + c_Q \cdot i(p) \\ = \; & c_Q \cdot (i(p+1) - 1) + c_A \cdot (i(p+1) - i(p) - 1) \end{aligned}$$

for each $0 \le p < P$. In steps 11–14, there are only non-q-steps. Processor 0 is the only processor performing the else-case for all $0 \le i < \lceil \log_t P \rceil$. In steps 13–14, there are at most $t-1$ non-q-steps in each lap which gives a total of $\lceil \log_t P \rceil \cdot (t-1)$ non-q-steps.

Each lap of the loop in Algorithm 10.4 step 11–14 causes at most $(t-1)$ copy steps. Including the final computation of processor 0, we have $\lceil \log_t P \rceil + 1$ supersteps. For the last step, we have no communication. Thus, the total communication costs are at most $\lceil \log_t P \rceil \cdot (\ell + g \cdot (t-1))$. □

An optimal partition. We have to specify step 2 of Algorithm 10.4. This is one starting point to minimize the overall cost. For a single processor or $\lambda_q(e) = 1$, there is nothing to do; hence, $\lambda_q(e) \ge P \ge 2$ in what follows. We have $\Lambda_p = i(p+1) - i(p)$ digits of $(e)_q$ that are assigned to processor p. If we assume the non-zero digits to be equal-distributed among all digits of $(e)_q$, then a digit is expected to be non-zero with probability $v = \frac{\nu_q(e)}{\lambda_q(e)} > 0$. Hence $v \cdot (\Lambda_p - 1)$ is just the number of (expected) non-q-steps to compute $E_p = \sum_{i(p)\le j<i(p+1)} e_j q^j$. For the worst case analysis, we set $v = 1$. Since $0 = i(0) < i(1) < \cdots < i(P) = \lambda_q(e)$ is a partition of $(e)_q$, we have

$$i(p+1) = \sum_{0\le p'\le p} \Lambda_{p'} \quad \text{and} \quad \lambda_q(e) = \sum_{0\le p<P} \Lambda_p. \tag{10.6}$$

But $\max_{0\le p<P}\{c_A \cdot v \cdot (i(p+1) - i(p) - 1) + c_Q \cdot (i(p+1) - 1)\}$ is minimal if all terms are equal:

$$\begin{aligned} & c_A \cdot v \cdot (\Lambda_p - 1) + c_Q \cdot \left(\sum_{0\le p'\le p} \Lambda_{p'} - 1 \right) \\ = \; & c_A \cdot v \cdot (\Lambda_{p-1} - 1) + c_Q \cdot \left(\sum_{0\le p'\le p-1} \Lambda_{p'} - 1 \right) \text{ for } 1 \le p < P. \end{aligned}$$

We set $c = \frac{c_Q}{c_A}$ and do some simplifications on the equation to get

$$\Lambda_p \cdot (v + c) = \Lambda_{p-1} \cdot v \text{ for } 1 \le p < P. \tag{10.7}$$

Since $v > 0$ and $c \ge 0$, we have $v + c > 0$. Inductively, we modify (10.7) to get

$$\Lambda_p = \left(\frac{v}{v+c} \right)^p \cdot \Lambda_0 \text{ for } 0 \le p < P. \tag{10.8}$$

Thus,

$$\lambda_q(e) = \sum_{0 \le p < P} \Lambda_p = \Lambda_0 \cdot \sum_{0 \le p < P} \left(\frac{v}{v+c} \right)^p ,$$

and we have to distinguish between two cases. For $c = 0$, this is

$$\lambda_q(e) = \Lambda_0 \cdot \sum_{0 \le p < P} 1^p = \Lambda_0 \cdot P, \tag{10.9}$$

and we have $\Lambda_0 = \cdots = \Lambda_{P-1} = \frac{\lambda_q(e)}{P}$ the *uniform partition of* $(e)_q$. This is the situation discussed in Stinson (1990) and von zur Gathen (1992). For $c > 0$, we have

$$\lambda_q(e) = \Lambda_0 \cdot \frac{\left(\frac{v}{v+c}\right)^P - 1}{\frac{v}{v+c} - 1} = \Lambda_0 \cdot \frac{v+c}{c} \cdot \left(1 - \left(\frac{v}{v+c} \right)^P \right) \tag{10.10}$$

with the help of the geometric series. We substitute Λ_0 by $\Lambda_{P-1} \cdot \left(\frac{v+c}{v}\right)^{P-1}$ due to (10.8), and we set $v = 1$. Then

$$\begin{aligned} \lambda_q(e) &= \Lambda_{P-1} \cdot (1+c)^{P-1} \cdot \frac{1+c}{(1+c)^P} \cdot \frac{1-(1+c)^P}{1-(1+c)} \\ &= \Lambda_{P-1} \cdot \frac{(1+c)^P - 1}{c} \end{aligned}$$

EXAMPLE 10.3 CONTINUED. The exponent 53 has binary length $\lambda_2(53) = 6$, and Hamming weight $\nu_2(53) = 4$, i.e. $v = \frac{\nu_2(53)}{\lambda_2(53)} = \frac{2}{3}$. The chain has weight $(1, 2)$, i.e. $c = \frac{1}{2}$. The formula above suggests the partition $(1|10|101)$ for $(53)_2$, since

$$\begin{array}{lllllll} \Lambda_0 & \approx & \frac{98}{31} & = & 3.16\ldots & \rightsquigarrow & \Lambda_0 = 3 \\ \Lambda_1 & \approx & \frac{56}{31} & = & 1.80\ldots & \rightsquigarrow & \Lambda_1 = 2 \\ \Lambda_2 & \approx & \frac{32}{31} & = & 1.03\ldots & \rightsquigarrow & \Lambda_2 = 1. \end{array}$$

This partition supports faster computation of an 2-addition chain with weight $(1, 2)$ for 53 than the uniform partition as shown in Figure 10.2. ◊

We substitute $\Lambda_{P-1} = i(P) - i(P-1)$ and $i(P) = \lambda_q(e)$ to get a bound on Algorithm 10.4 step 2:

PROPOSITION 10.11. *Let e be an exponent, q be a scalar greater than 1, and P with $1 \le P \le \lambda_q(e)$ be the number of processors. Then there are a partition for*

$(e)_q$ and a q-addition chain γ with weight (c_Q, c_A) that computes the exponents of this partition using at most P processors in depth

$$\delta_{q,(c_Q,c_A)}(\gamma) \leq \begin{cases} c_A \cdot \left\lceil \frac{\lambda_q(e)}{P} \right\rceil & \text{if } c_Q = 0, \text{ and} \\ c_Q \cdot (\lambda_q(e) - 1) + c_A \cdot \left\lceil \lambda_q(e) \cdot \frac{c}{(1+c)^P - 1} - 1 \right\rceil & \text{if } c_Q > 0 \end{cases}$$

if the nodes $\{(a,p)\colon 1 \leq a < q, 0 \leq p < P\}$ are already precomputed.

For $c_Q > 0$, we observe that the processor labeled $P-1$ mainly performs q-steps. This is the bottleneck of the parallel computation as identified in Section 4.1.

The optimal tree. The second tuning parameter is given by $t \in \mathbb{N}_{\geq 2}$. This parameter determines the depth of the collection stage in Algorithm 10.4 step 11–14. It fixes the number of inner nodes in the communication tree. The previously determined partition of e does not depend on t. Therefore, we can handle the collection separately to further decrease the overall cost. We have to minimize

$$c_A \cdot \lceil \log_t P \rceil \cdot (t-1) + \lceil \log_t P \rceil (\ell + g \cdot (t-1)).$$

The first summand counts the number of non-q-steps, the second one describes the communication cost. For simplicity, we approximate $\lceil \log_t P \rceil$ by $\frac{\ln P}{\ln t}$ and discuss the differentiable function $L\colon \mathbb{R}_{\geq 2} \to \mathbb{R}$ with

$$L(t) = c_A \cdot \frac{\ln P}{\ln t} \cdot (t-1) + \frac{\ln P}{\ln t} \cdot (\ell + g \cdot (t-1)) = \frac{t}{\ln t} \cdot u + \frac{v}{\ln t}$$

where $u = \ln P \cdot (c_A + g)$ and $v = \ln P \cdot (\ell - c_A - g)$. The first derivation is

$$L'(t) = \frac{t \cdot (\ln t - 1) \cdot u - v}{t \ln^2 t},$$

and a local minimum at t_0 with $t_0 \ln^2 t_0 \neq 0$ satisfies

$$t_0 \cdot (\ln t_0 - 1) \cdot u - v = 0,$$

which is equivalent to

$$\exp(\ln t_0 - 1) \cdot (\ln t_0 - 1) = \frac{v}{u} \exp(-1).$$

This is an equation of type $w \cdot \exp(w) = x$ which is given by *Lambert's W function*, see page 22. Thus, we have

$$\ln t_0 - 1 = W\left(\frac{v}{u} \cdot \exp(-1)\right)$$

which has the solution

$$t_0 = \exp\left(W\left(\frac{v}{u} \cdot \exp(-1)\right) + 1\right).$$

This is indeed the global minimum. If we neglect latency, i.e. $\ell = 0$, then $u = \ln P \cdot (c_A + g) = -v$ and $t_0 = \exp(W(-1 \cdot \exp(-1)) + 1) = \exp(-1 + 1) = 1$ since $W(-1 \cdot \exp(-1)) = -1$. The best integer value for $t \in \mathbb{N}_{\geq 2}$ is then $t_0 = 2$ since $L'(t) > 0$ for $t \geq 2$. We summarize our discussion for the main stage inserting Proposition 10.11 and a binary tree in Lemma 10.5.

COROLLARY 10.12. *Let e be an exponent, q be a scalar greater than 1, and P be the maximal number of processors with $1 \leq P \leq \lambda_q(e) = \lambda$. Let γ' be an addition chain with $\mathcal{S}(\gamma') \times \{0, \ldots, P-1\} \supseteq \{(a,p) \colon a \in \mathcal{E}, 0 \leq p < P\}$ for $\mathcal{E} = \{1, \ldots, q-1\}$.*

(i) There is a q-addition chain γ with weight (c_Q, c_A) that computes e using P processors in depth

$$\delta_{q,(c_Q,c_A)}(\gamma) \leq \delta_{q,(c_Q,c_A)}(\gamma') + \begin{cases} c_A \cdot \left(\left\lceil \frac{\lambda}{P} \right\rceil - 1 + \lceil \log_2 P \rceil\right) & \text{if } c_Q = 0, \text{ and} \\ c_Q \cdot (\lambda - 1) + c_A \cdot \lceil \log_2 P \rceil & \\ \quad + c_A \cdot \left\lceil \frac{c}{(1+c)^P - 1} \cdot \lambda - 1 \right\rceil & \text{if } c_Q > 0. \end{cases}$$

(ii) In the BSP-model, this can be implemented with $S = \lceil \log_2 P \rceil + 1$ supersteps and communication cost

$$C_{\ell,g}(\gamma) \leq \lceil \log_2 P \rceil \cdot (\ell + g)\,.$$

10.3. Parallel precomputation. The main computation needs a precomputation stage whenever $q > 2$. We recall Brauer's idea which we have presented in Section 3.2, and introduce a tuning parameter $m \in \mathbb{N}_{\geq 1}$. It reduces the total depth by balancing the costs for precomputation and the main stage. The precomputation generates a q-addition chain γ' for $\mathcal{S}(\gamma') = \{1, 2, \ldots, q^m - 1\}$. The main stage computes a q^m-addition chain γ for a given exponent $e \in \mathbb{N}_{\geq 1}$. We take the communication cost into account and recall that the number of processors is limited. The number of elements that can be calculated in a few parallel steps is restricted by Theorem 4.22. This does not depend on the number of available processors. The limited resources become a bottleneck for the precomputation whenever the order of $\mathcal{S}(\gamma')$ is large compared to P. Thus, we divide the precomputation into two successive parts: an initial one and a flowing one. The resulting algorithm will prove the following bound:

LEMMA 10.13. *Let q be a scalar greater than 1, m be a positive integer, and (c_Q, c_A) be a weight.*

(i) Using at most $P \in \mathbb{N}_{\geq 1}$ processors there is a q-addition chain γ' with weight

(c_Q, c_A) computing $\mathcal{E} = \{1, \ldots, q^m - 1\}$ in depth

$$\delta_{q,(c_Q,c_A)}(\gamma') \leq \begin{cases} c_A \cdot \lceil m \cdot \log_2 q \rceil & \text{if } P' = 2^{\lceil \log_2 P \rceil} + P \\ & \geq q^m - 1, \text{ and} \\ c_A \cdot \left(\frac{1}{P} \cdot (q^m - q^{m-1}) + 1 + \lceil \log_2 P \rceil\right) & \\ + c_Q \cdot \left(\frac{1}{P} \cdot \frac{q^m - 1}{q-1} + \frac{1}{2} \cdot (m^2 + 3m)\right) & \text{otherwise.} \end{cases}$$

(ii) In the BSP-model, this can be implemented with

$$S \leq \begin{cases} \lceil m \cdot \log_2 q \rceil & \text{if } P' \geq q^m - 1, \text{ and} \\ \lceil \log_2 P \rceil + 2 & \text{else} \end{cases}$$

supersteps and communication cost at most

$$C_{\ell,g}(\gamma) \leq \begin{cases} \ell \cdot \lceil m \cdot \log_2 q \rceil + g \cdot \left(\frac{3}{4} q^{2m} - 2q^m + 1\right) & \text{if } P' \geq q^m - 1, \\ \ell \cdot (\lceil \log_2 P \rceil + 2) + g \cdot \left(q^m \cdot (1 - \frac{1}{P}) \right. & \text{otherwise.} \\ \left. + 3P^2 - 4P + 10 + \frac{1}{2} \cdot (m^2 + m)\right) & \end{cases}$$

We illustrate the dependence of the precomputation on the number of processors by an example.

EXAMPLE 10.14. Let $q = 2$ and $m = 3$. We assume $c_A = c_Q = 1$, and therefore the edges are not labeled. The computation of γ' for $\mathcal{E} = \{1, 2, 3, 4, 5, 6, 7\}$ may be as in Figure 10.3. For $P = 1$ the number of processors is the bottleneck. This is no longer true for $P \geq 3$ processors. In the latter case, the depth is a consequence of the lower bound on parallel exponentiation. ◊

For the communication costs, we assume that all processors have stored all elements $\{1, \ldots, q^m - 1\}$ after the precomputation locally, i.e. $(a, p) \in \mathcal{S}(\delta) \times \mathcal{P}$ for all $1 \leq a < q^m$ and $0 \leq p < P$.

Initialization. In the first part of the precomputation, we calculate and distribute nodes until each processor has computed at least one node. Thus, we compute a total of $P' = 2^{\lceil \log_2 P \rceil} + P$ exponents if $P \in \mathbb{N}_{\geq 2}$ processors are available. Without loss of generality, we restrict the number of processors and assume $P' \leq q^m - 1$; the number of processors *is not* a bottleneck for the initialization stage.

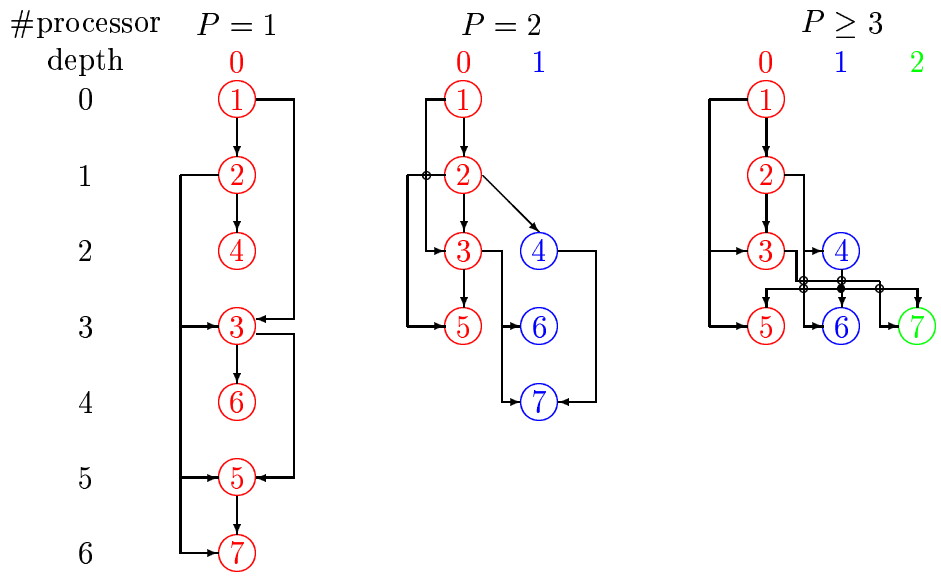


Figure 10.3: An illustration of the precomputation stage when the number of processors changes. The parameters are described in Example 10.14.

ALGORITHM 10.15. Precomputation: initialization.

Input: A scalar $q \in \mathbb{N}_{\geq 2}$, and the number of processors $P \in \mathbb{N}_{\geq 1}$.
Output: A q-addition chain γ' computing the set $\mathcal{E} = \{1, 2, \ldots, P'\}$ where $P' = 2^{\lceil \log_2 P \rceil} + P$. The q-addition chain satisfies $\mathcal{E} \times \{0, \ldots, P-1\} \subseteq \mathcal{S}(\gamma') \times \{0, \ldots, P-1\}$.

1. Set γ' the empty addition chain with $\mathcal{S}(\gamma') = \{1\}$.
2. For all processors $0 \leq p < P$ in parallel do 3–7
3. For $0 \leq i \leq \lceil \log_2 P \rceil$ do 4–7
4. If $p < 2^i$ then
5. Set $e = 2^i + (p+1)$.
6. Add node (e, p) and rule $((2^i, p), (p+1, p))$ to γ'.
7. For all $0 \leq p' < P$ with $p \neq p'$ do add the node (e, p') to γ'.
8. Return γ'.

PROPOSITION 10.16. *The algorithm works as specified.*

(i) The q-addition chain γ' with weight (c_Q, c_A) computing $\mathcal{E} = \{1, \ldots, P'\}$ can be performed in depth

$$\delta_{q,(c_Q,c_A)}(\gamma') \leq c_A \cdot (\lceil \log_2 P \rceil + 1) .$$

(ii) In the BSP-model this can be implemented with $S = \lceil \log_2 P \rceil + 1$ *supersteps and communication cost at most*

$$C_{\ell,g}(\gamma') \leq \ell \cdot (\lceil \log_2 P \rceil + 1) + g \cdot (P' - 1) \cdot (P - 1).$$

PROOF. Correctness follows with the invariant: After lap i of steps 3–7 all nodes (a, p) with $1 \leq a \leq \min\{2^{i+1}, P'\}$ and $0 \leq p < P$ are in γ'. We show this by induction on i.

Before the loop, all processors have generated the empty addition chain and thus $\mathcal{S}(\gamma') \times \{0, \ldots, P-1\} = \{(1, p) \colon 0 \leq p < P\}$. Now, we suppose that the invariant holds before lap i. Let e be an element with $2^i < e \leq \min\{2^{i+1}, P'\}$. By the induction hypothesis, the nodes $(2^i, p)$ and $(p+1, p)$ are already in γ' since $0 \leq p < 2^i$. Therefore, processor $p = e - 2^i - 1 < \min\{2^i, P\}$ computes node (e, p) correctly. Step 7 distributes e to all processors. Thus, after the lap any processor $0 \leq p < P$ has stored the nodes (a, p) with $1 \leq a \leq \min\{2^{i+1}, P'\}$.

There are only non-q-steps. Only processor 0 computes a node in every lap of steps 3–7. This proves the claimed depth.

In the BSP-model, we can identify each lap by a superstep, i.e. $S = \lceil \log_2 P \rceil + 1$. In lap $0 \leq i \leq \lceil \log_2 P \rceil$ each processor $0 \leq p < \min\{2^i, P\}$ sends $P - 1$ copies of its result to all other processors. This causes a total communication of

$$\begin{aligned}
&\sum_{0 \leq i \leq \lceil \log_2 P \rceil} (P-1) \cdot \min\{2^i, P\} = (P-1) \cdot \left(P + \sum_{0 \leq i < \lceil \log_2 P \rceil} 2^i \right) \\
&= P^2 - P + (P-1) \cdot \left(2^{\lceil \log_2 P \rceil} - 1 \right) \\
&= P \cdot \left(P + 2^{\lceil \log_2 P \rceil} \right) - \left(P + 2^{\lceil \log_2 P \rceil} \right) - (P-1) \\
&= P \cdot P' - P' - (P-1) = (P'-1) \cdot (P-1)
\end{aligned}$$

as claimed. □

The flow. The initialization generates a chain of at most $P' = 2^{\lceil \log_2 P \rceil} + P \geq 2P$ elements. If $q = 2$ and $m = 3$ then the precomputation is complete whenever at least 3 processors are involved, as illustrated in Example 10.14. For larger values of m or q, or a smaller number of processors, we have to add a second stage. This is a little bit difficult, since we want to avoid unnecessary communication. We assume that all processors communicate their locally stored intermediate results to all other processors. Thus, we focus on the reduction of the number of supersteps and, hence, the total start up cost $S \cdot \ell$.

We define the sequence of all positive integers $b_0, b_1, \ldots$ not divisible by a given scalar $q \in \mathbb{N}_{\geq 2}$ by

$$b_0 = 1 \quad \text{and} \quad b_i = \begin{cases} b_{i-1} + 2 & \text{if } q \text{ divides } b_{i-1} + 1, \\ b_{i-1} + 1 & \text{else} \end{cases} \quad \text{for } i \in \mathbb{N}_{\geq 0}. \tag{10.17}$$

We have $b_{i+P} - b_i \le (b_i + 2P) - b_i = 2P$ for all $i \in \mathbb{N}_{\ge 0}$, and all involved processors have stored the first $P' \ge 2P$ elements $1, \ldots, P'$ locally. Thus, each processor $0 \le p < P$ can compute the nodes $b_{i+Pj} > P'$ for an $i \le P'$ and $j \in \mathbb{N}_{\ge 1}$ without falling back upon non-locally stored predecessors. All missing exponents are multiples of q and can be calculated subsequently. We formalize the described second part of the precomputation.

ALGORITHM 10.18. Precomputation: flow.

Input: A scalar $q \in \mathbb{N}_{\ge 2}$, a set of processors $0 \le p < P$, a parameter $m \in \mathbb{N}_{\ge 1}$ such that $P' = 2^{\lceil \log_2 P \rceil} + P < q^m - 1$, and a q-addition chain γ' with $\{(a,p) : 1 \le a \le P', 0 \le p < P\} = \mathcal{S}(\gamma') \times \{0, \ldots, P-1\}$.

Output: A q-addition chain γ computing $\mathcal{E} = \{1, \ldots, q^m - 1\}$.

1. Set $\gamma = \gamma'$.
2. For all processors $0 \le p < P$ in parallel do 3–12
3. Set $a = b_p$ and $j = 0$.
4. While $a < q^m$ do 5–10
5. Set $b \leftarrow a \cdot q$.
6. While $b < q^m$ do 7–8
7. If $b > P'$ then add node (b,p) and rule $((\frac{b}{q}, p), -q)$ to γ.
8. Set $b \leftarrow b \cdot q$.
9. Set $j \leftarrow j + 1$ and $a = b_{p+jP}$.
10. If $a < q^m$ and $a > P'$ then add rule $((a - b_{p+(j-1)P}, p), (b_{p+(j-1)P}, p))$ and node (a,p) to γ.
11. For all $(a,p) \in \mathcal{S}(\gamma) \times \{0, \ldots, P-1\}$ with $P' < a < q^m$ do
12. For $0 \le p' < P$ with $p' \ne p$ do add node (a,p') to γ.
13. Return γ.

PROPOSITION 10.19. *The algorithm works as specified.*

(i) The q-addition chain γ with weight (c_Q, c_A) computing $\mathcal{E} = \{1, \ldots, q^m - 1\}$ can be generated in depth

$$\begin{aligned} \delta_{q,(c_Q,c_A)}(\gamma) \;\le\; & \delta_{q,(c_Q,c_A)}(\gamma') + c_A \cdot \frac{1}{P} \cdot \left(q^m - q^{m-1}\right) \\ & + c_Q \cdot \left(\frac{1}{P} \cdot \frac{q^m - 1}{q-1} + \frac{P-1}{2P} \cdot (m^2 + 3m) \right). \end{aligned}$$

(ii) In the BSP-model, this can be implemented with $S = 1$ supersteps and communication cost of at most

$$C_{\ell,g}(\gamma) \;\le\; C_{\ell,g}(\gamma') + \ell + g \cdot q^m \cdot (1 - \frac{1}{P}) + g \cdot \left(\frac{m^2 + m}{2} + 9 \right).$$

Before we give the proof, we do some preparation. For the cost analysis we count the number of non-q-steps and the number of q-steps, which are added by each processor $0 \leq p < P$, separately. Thus, we group the multiples of q in the set $\mathcal{E} = \{1, \ldots, q^m - 1\}$. The multiples $1 \cdot q, \ldots, 1 \cdot q^{m-1}$ of q are discounted here.

REMARK 10.20. *Let $1 \leq t < m$ be an integer. Then there are exactly $q^t - q^{t-1} - 1$ elements $a \in \{2 \leq a' < q^m : q \text{ does not divide } a'\}$ such that $\#\{i \in \mathbb{N}_{\geq 1} : aq^i < q^m\} = m - t$.*

PROOF. Let $a \in \{2, \ldots, q^m - 1\}$ such that $q^{t-1} + 1 \leq a < q^t$ for a fixed $t \in \mathbb{N}_{\geq 1}$. Then we have $q^m > aq^i > q^t q^i = q^{t+i}$ for a if and only if $i \leq m - t$. Since $(q^{t-1}+1) \cdot q^{m-t} = q^{m-1} + q^{m-t} \leq a < q^t q^{m-t} = q^m$, there are exactly $q^t - (q^{t-1}+1)$ elements a not divisible by q which satisfy $\#\{i \in \mathbb{N}_{\geq 1} : aq^i < q^m\} = m - t$. □

PROOF (of Proposition 10.19). Correctness can be seen as follows: All processors $0 \leq p < P$ have stored the nodes $(1, p), \ldots, (P', p)$ as an input in step 1. Processor p starts with $(b_p, p) \in \mathcal{S}(\gamma)$ in step 3. Since $b_p \leq 2P \leq P'$ this node is stored locally. Hence, in steps 9–10 processor p can calculate b_{p+jP} as claimed in step 10 of lap j using only elements that have been stored in local memory. Inductively, all elements $b_{p+jP} \cdot q^i < q^m$ are computed and stored in step 6–8 by processor p. Thus in step 12, processor p copies all locally generated elements to all other processors, and we have $\{(a, p) : 1 \leq a < q^m, p \in \{0, \ldots, P-1\}\} \subseteq \mathcal{S}(\gamma) \times \{0, \ldots, P-1\}$ as claimed.

For the cost analysis, we count the non-q-steps first. Exactly $q^m - 2 - \lfloor \frac{q^m - 1}{q} \rfloor = q^m - q^{m-1} - 1$ non-q-steps have to be generated in the precomputation. Exactly $P' < q^m - 1$ elements are computed by γ', divided into $\lfloor \frac{P'}{q} \rfloor$ many q-steps and $P' - \lfloor \frac{P'}{q} \rfloor$ non-q-steps. The remaining

$$A = q^m - q^{m-1} - 1 - \left(P' - \left\lfloor \frac{P'}{q} \right\rfloor \right)$$

non-q-steps are equally distributed to the P processors. Thus, each processor has to compute at most $\lceil \frac{A}{P} \rceil$ many of them. We insert $P' - \lfloor \frac{P'}{q} \rfloor \geq P' - \frac{P'}{q} \geq 2P \cdot \frac{q-1}{q}$ to get

$$\left\lceil \frac{A}{P} \right\rceil < q^{m-1} \cdot \frac{q-1}{P} - 2 \cdot \frac{q-1}{q} - \frac{1}{P} \leq q^{m-1} \cdot \frac{q-1}{P}.$$

For the q-steps, we suppose that no multiples of q are in γ'. Hence, our upper bound on the number of q-steps may be too generous. Then processor $0 \leq p < P$ adds at most t many q-steps in step 6–8 of Algorithm 10.18 to γ for each a not divisible by q. By Remark 10.20, there are $q^t - q^{t-1} - 1$ many elements with t

many q-steps. The number of q-steps for each processor is at most

$$\begin{aligned}
\left\lceil \frac{Q}{P} \right\rceil &\leq \left\lceil \sum_{1\leq t\leq m} \left\lceil \frac{q^t - q^{t-1} - 1}{P} \right\rceil \cdot (m-t) \right\rceil \\
&< \sum_{1\leq t\leq m} \left(\frac{q^t - q^{t-1} - 1}{P} + 1 \right) \cdot (m-t) \\
&= \frac{q-1}{P} \left(m \cdot \sum_{1\leq t\leq m} q^{t-1} - \sum_{1\leq t\leq m} t q^{t-1} \right) + \frac{P-1}{P} \cdot \left(\sum_{1\leq t\leq m} (m-t) \right) \\
&= \frac{q-1}{P} \cdot \left(m \cdot \frac{q^m - 1}{q-1} - m \cdot \frac{q^m (q-1)}{(q-1)^2} + \frac{q^m - 1}{(q-1)^2} \right) + \frac{P-1}{P} \cdot \frac{m^2 + m}{2} \\
&= \frac{q^m - 1}{P \cdot (q-1)} + \frac{P-1}{2P} \cdot (m^2 + m) - \frac{m}{P} \\
&= \frac{1}{P} \cdot \frac{q^m - 1}{q-1} + \frac{1}{2} \cdot (m^2 + m) - \frac{1}{2P} \cdot (m^2 + 3m).
\end{aligned}$$

Furthermore, processor 0 has to compute the elements $q, q^2, \ldots, q^{m-1}$. For the communication cost we observe that all processors copy the locally stored elements to all other processors. Each processor receives all elements but the precomputed P' elements and the elements it has computed on its own, i.e.

$$q^m - 1 - P' - \left\lfloor \frac{A}{P} \right\rfloor - \left\lfloor \frac{Q}{P} \right\rfloor$$

many elements. We have

$$\begin{aligned}
\left\lfloor \frac{A}{P} \right\rfloor &\geq \frac{1}{P} \cdot (q^m - q^{m-1} - 1 - P' \cdot (1 - \frac{1}{q}) - 1) - 1 \\
&\geq \frac{q-1}{P} \cdot q^{m-1} - \frac{2}{P} - \frac{3}{P} + \frac{3}{q} - 1 \\
&\geq \frac{q-1}{P} \cdot q^{m-1} - 6
\end{aligned}$$

and

$$\begin{aligned}
\left\lfloor \frac{Q}{P} \right\rfloor &\geq \sum_{1\leq t\leq m} \left\lfloor \frac{q^t - q^{t-1} - 1}{P} \right\rfloor \cdot (m-t) - \left\lceil \frac{P'}{q} \right\rceil \\
&\geq \sum_{1\leq t\leq m} \left(\frac{q^t - q^{t-1} - 1}{P} - 1 \right) \cdot (m-t) - \frac{3}{q} - 1 \\
&\geq \frac{1}{P} \cdot \frac{q^m - 1}{q-1} - \frac{m^2 + m}{2} + \frac{m^2 - m}{2P} - 4.
\end{aligned}$$

Thus, we have

$$C_{\ell,g}(\gamma) \leq \ell + g \cdot \left(q^m \cdot \left(1 - \frac{1}{P} \cdot \frac{q^2 - q + 1}{q^2 - q} \right) + 9 + \frac{m^2 + m}{2} - \frac{m^2 - m}{2P} \right)$$

as a bound on the communication cost. □

PROOF (of Lemma 10.13). If we have $q^m - 1 \leq P' = 2^{\lceil \log_2 P \rceil} + P$ then there is only an initialization part. At most $P = \lceil \frac{q^m - 1}{2} \rceil \geq \frac{q^m - 1}{2}$ processors compute $P' = 2^{\lceil \log_2 P \rceil} + P \geq 2 \cdot \frac{q^m - 1}{2} = q^m - 1$ elements. Thus, $P = \lceil \frac{q^m - 1}{2} \rceil$ processors are sufficient to compute $q^m - 1$. We insert this value for P in Proposition 10.16 to get the estimate; in this case using $\lceil \log_2 P \rceil + 1 \leq \lceil \log_2 (q^m) \rceil - 1 + 1 \leq \lceil m \cdot \log_2 q \rceil$.

If $q^m - 1 > P'$ then the estimates of the initialization in Proposition 10.16 and the flow part in Proposition 10.19 have to be summed up. We set $P' < 3P$ to get the claimed upper bound. □

Connecting precomputation and main stage. It remains to connect the precomputation and the main stage. We restrict to exponentiation in the finite field $\mathbb{F}_{q^n}$. By Fermat's Little Theorem 2.3, we have $0 \leq \lambda_q(e) < n$. The proof of Result 10.1 follows with Corollary 10.12 and Lemma 10.13, inserting a suitable value for the connecting parameter $m \in \mathbb{N}_{\geq 1}$. We suppose that the field extension is large enough, i.e. $\frac{n}{\ln^2 n} \geq \frac{1}{\ln^2 q}$. As in the proof of Corollary 3.10, we choose $m = \lfloor \log_q n - 2 \log_q \log_q n \rfloor + 1 \geq 1$, if the number of processors P is small compared to n, i.e. $\lceil \frac{n}{P} \rceil \leq \lfloor \log_q n - 2 \log_q \log_q n \rfloor + 1$. If there are more processors available then we set $m = \lceil \frac{n}{P} \rceil \geq 1$.

We first discuss the case $m = \lfloor \log_q n - 2 \log_q \log_q n \rfloor + 1 \geq 1$. Then the bound on the depth of the precomputation stage in Lemma 10.13 is as follows. For $3P > P' = 2^{\lceil \log_2 P \rceil} + P \geq q^m - 1$, we have

$$\delta_{q,(c_Q,c_A)}(\gamma') \leq c_A \cdot (\lceil \log_2 (3P) \rceil + 1) \leq c_A \cdot (\lceil \log_2 P \rceil + 3). \tag{10.21}$$

If the number of processors P is smaller, then the precomputation has two parts. In this case, the depth is bounded by

$$\begin{aligned} \delta_{q,(c_Q,c_A)}(\gamma') \leq\ & c_A \cdot (\lceil \log_2 P \rceil + 1) + c_Q \cdot O(\log^2 n) \\ & + c_A \cdot \frac{1}{P} \cdot (q - 1) \cdot \frac{n}{\log_q^2 n} \\ & + c_Q \cdot \frac{1}{P} \cdot \frac{q}{q - 1} \cdot \frac{n}{\log_q^2 n}. \end{aligned} \tag{10.22}$$

If the number of processors P is large relative to n, then we choose $m = \lceil \frac{n}{P} \rceil$. The bound does not change in the case $P' \geq q^m - 1$. Since $m \leq \lfloor \log_q n - 2 \log_q \log_q n \rfloor + 1$, the previous estimate is also an upper bound on the depth if P is large compared

to n. Therefore, we always have basic cost at most $c_A \cdot (\lceil \log_2 P \rceil + 3)$ for the initialization. If there is a successive flow part in the precomputation then the work of this part is (nearly) evenly distributed among the P processors.

The estimate on the depth of the main stage is given by Corollary 10.12. The main stage works with the q^m-ary representation of the exponent. We have $\lambda_{q^m}(e) \leq \frac{1}{m} \cdot \lambda_q(e) + 1 \leq \frac{n}{m} + 1$ for all $0 \leq e < q^n$.

For $m = \lceil \frac{n}{P} \rceil$, the partition is uniform for both $c_Q = 0$ and $c_Q > 0$. Each processor $0 \leq p < P$ has to compute only q-steps to calculate its intermediate result $E_p = e_p q^p$ from the precomputed digit e_p. This takes at most $c_Q \cdot m \cdot (\lambda_{q^m}(e) - 1) \leq c_Q \cdot \lambda_q(e)$ many q-steps. Our scalable algorithm is also powerful if $m = \lfloor \log_q n - 2 \log_q \log_q n \rfloor + 1$. Then the intermediate results E_p computed by processors $0 \leq p < P$ may contain sums of digits. For $c_Q = 0$, the depth is bounded with respect to Corollary 10.12. We have $\lambda_{q^m}(e) \leq \frac{n}{m} + 1$

$$
\begin{aligned}
(10.23) \qquad & \delta_{q,(0,c_A)}(\gamma) - \delta_{q,(0,c_A)}(\gamma') \leq c_A \cdot \left(\frac{1}{P} \cdot \left(\frac{n}{m} + 1 \right) + \lceil \log_2 P \rceil \right) \\
& \leq c_A \cdot \frac{n}{\log_q n} \cdot \frac{1}{P} \cdot \left(1 + \frac{2 \log_q \log_q n}{\log_q n - 2 \log_q \log_q n} \right) + c_A \cdot (\lceil \log_2 P \rceil + 1) \\
& \leq c_A \cdot \frac{1}{P} \cdot \frac{n}{\log_q n} \cdot (1 + o(1)) + c_A \cdot (\lceil \log_2 P \rceil + 1) \, .
\end{aligned}
$$

The exponentiation problem is highly scalable, because the non-q-steps can be evenly distributed to all available processors. The only additional cost is given by the collection. For small values of P, we expect a good speed-up if $c_Q = 0$. Thus, this favorable case offers not only the highest parallel speed-up for parallel exponentiation but also good scalability. For $c_Q > 0$, the main stage reveals the bottleneck of parallel exponentiation. In this case, we suppose that the cost for a q^m-step are (roughly) the costs for m many q-steps. Then

$$
\begin{aligned}
& \delta_{q,(0,c_A)}(\gamma) - \delta_{q,(0,c_A)}(\gamma') \\
\leq \; & c_Q \cdot (n-1) + c_A \cdot \lceil \log_2 P \rceil \\
& + c_A \cdot \left\lceil \frac{c}{(1+c)^P - 1} \cdot \frac{n}{\log_q n - 2 \log_q \log_q n} \right\rceil \\
(10.24) \qquad \leq \; & c_Q \cdot (n-1) + c_A \cdot (\lceil \log_2 P \rceil + 1) \\
& + c_A \cdot \frac{c}{(1+c)^P - 1} \cdot \frac{n}{\log_q n} \cdot (1 + o(1)).
\end{aligned}
$$

The number of q-steps is not touched by the number of processors P. Only the number of non-q-steps decreases if the number of available processors P increases. If $c = \frac{c_Q}{c_A}$ is close to 0, then we have $\frac{c}{(1+c)^P - 1}$ close to $\frac{1}{P}$, since $\lim_{c \to 0} \frac{c}{(1+c)^P - 1} = \lim_{c \to 0} \frac{1}{P \cdot 1} = \frac{1}{P}$ by de l'Hospital's rule. Therefore, we expect a high speed-up only for small amounts of P and $c_Q \ll c_A$. If the number of processors increases, the

bottleneck becomes dominant. The speed-up is limited by the number of q-steps which have to be done in sequential.

We merge all the estimates for small P given above to prove Result 10.1. The estimates for larger P can be found in a similar way.

PROOF (of Result 10.1). For $c_Q = 0$ we add (10.21) to (10.23) which gives

$$\delta_{q,(0,c_A)}(\gamma) \quad \leq \quad c_A \cdot (2 \cdot \lceil \log_2 P \rceil + 4) + c_A \cdot \frac{1}{P} \cdot \frac{n}{\log_q n} \cdot (1 + o(1)).$$

This estimate is also true if we have an additional flow part in the precomputation given by (10.22).

If $c_Q > 0$ and $P \geq \frac{n}{3 \log_q^2 n}$ then the depth of the precomputation is bounded by (10.21). Otherwise, we insert (10.22) in (10.24) to get the claimed upper bound on the depth. □

10.4. Experiments. It remains to summarize all our ideas presented throughout this work. The concluding experiments that are described now should verify our theoretical results on exponentiation in finite fields $\mathbb{F}_{2^n}$. We implemented the scalable parallel exponentiation algorithm that has been developed in this section. We included the arithmetic for the different basis representations of $\mathbb{F}_{2^n}$ that are discussed in the previous sections. Thus, these final experiments combine the algorithmic ideas on weighted 2-addition chains, on fast arithmetic in $\mathbb{F}_{2^n}$, and on parallel (scalable) exponentiation. The intention of these experiments is to validate the significance of the ratio $c = \frac{c_Q}{c_A}$ as the basic criterion to find suitable data structures for parallel exponentiation. They also should illustrate that the limits given by the lower bound in Section 4.2 are realizable in practice!

There are (at least) two further aspects that we want to illustrate by the experiments: We want to show that scalability really works for parallel exponentiation. And last but not least the tests should determine the breakpoint, i.e. the number of processors for which a normal basis representation given by a prime Gauß period of type $(n, 2)$ beats the polynomial basis representation with sparse modulus.

Experimental setup. We ran our implementation of Algorithm 10.4 with the four already introduced test series: *Arbitrary*, *Sparse*, *Sedimentary*, and *PrimeGP*. The test series *Normal* was not part of these final experiments because of the experimental and theoretical comparisons between this test series and the test series *PrimeGP*. Both test series coincide with respect to $c = 0$. The only difference between *Normal* and *PrimeGP* is given by the faster multiplication for the latter one. Our experiments on inversion in a normal basis representation (see Section 6.5 and Section 8.2) validated this observation; the structure of the computation was not touched, but the faster multiplication for prime Gauß periods sped up the whole computation significantly.

For each of the four test series we ran the exponentiation algorithm on 2, 4, 8, 16, and 32 processors in parallel. The programs were executed on the *PSC2* cluster of the *Paderborn Center for Parallel Computing*. As before, each test series contained 50 different extension fields $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$; we chose $n \approx 200i$ with $1 \leq i \leq 50$. For each value of n we did 100 trials for the three test series *Sparse*, *Sedimentary* and *PrimeGP*. For *Arbitrary* there were only 10 trials. For each trial an exponent of exactly n bits and an element of $\mathbb{F}_{2^n}$ were chosen at random. All times in Tables A.20–A.23 are average values for all trials with given n. To run a single test series in sequential took about 24 hours on the *PSC2*.

Evaluation. We compared the results on parallel exponentiation with two different sequential exponentiation algorithms: an implementation of Algorithm 3.12 (Brauer 2-addition chains) and of repeated squaring (Algorithm 3.12 with $q = 2$ and $m = 1$). The speed-up shown in Figures 10.4–10.7 is with respect to Brauer 2-addition chains. Since Brauer 2-addition chains are among the shortest known addition chains, the speed-up illustrates the profit that can be taken from parallel computing for exponentiation in $\mathbb{F}_{2^n}$ using different basis representations. We recall that Brauer's idea has also been our starting point to parallelize exponentiation; Algorithm 10.4 can be viewed as Algorithm 3.12 if we have only a single processor.

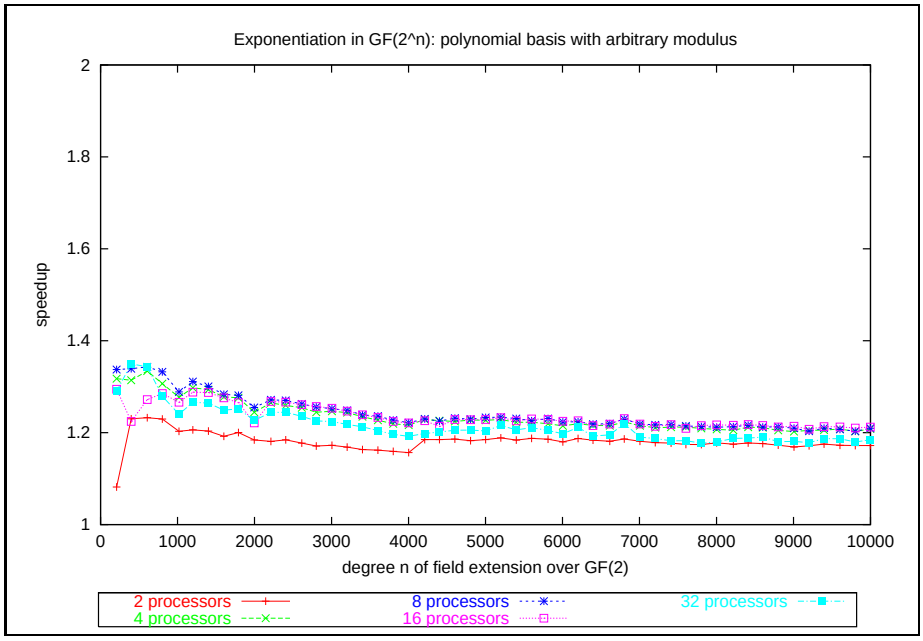


Figure 10.4: The speed-up of parallel exponentiation for the test series *Arbitrary* on the PSC2-cluster using 2, 4, 8, 16 and 32 processors. These speed-ups are relative to a sequential implementation of Algorithm 3.12.

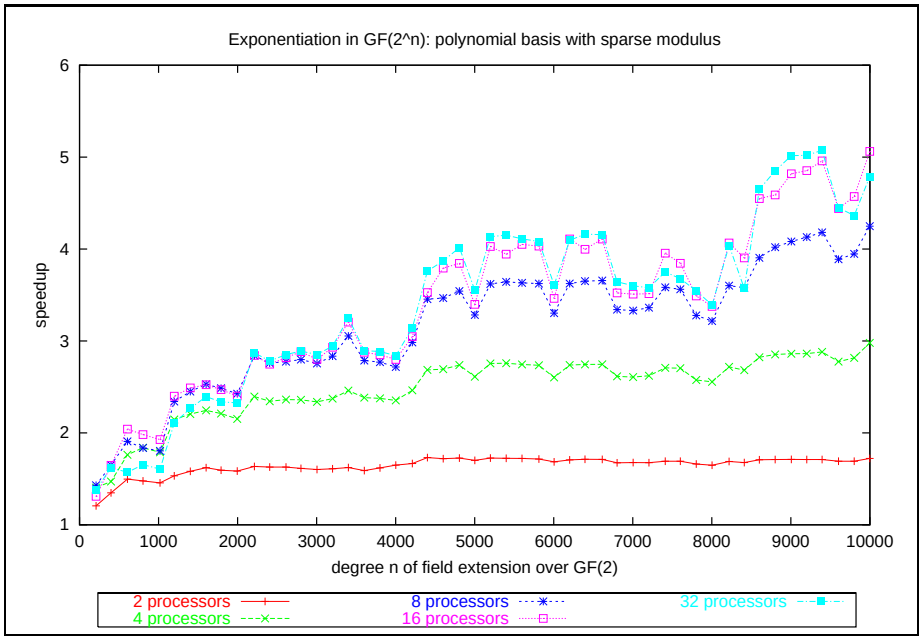


Figure 10.5: The same experiments as illustrated in Figure 10.5 but for the test series *Sparse*.

We compared parallel exponentiation to repeated squaring, see Table A.24 and Figure 10.8, respectively. The expected speed-up of Algorithm 4.12 with respect to repeated squaring has been documented in Figure 4.10. Those results base on theoretical estimates and are in a sense maximal speed-ups. The speed-ups illustrated in Figure 10.8 purely base on experiments and connect our experimental results to the theory. In particular, we can compare the two different approaches of Algorithm 4.12 and Algorithm 10.4 to each other. The latter one takes communication cost and a limited number of processors into account.

Results. The speed-ups for test series *Arbitrary* are documented in Figure 10.4 and Table A.20, respectively. The poor speed-up and the missing improvement for more than 4 processors are obvious. Both facts have been predicted by the theoretical results. Since $c \approx \frac{2}{3}$ is close to the assumptions of original addition chains, there is no real profit using parallel computing. Furthermore, three processors are sufficient to get the maximal speed-up due to Algorithm 4.12. This saturation coincides with the fact that—compared to 4 processors—only marginal better speed-ups are shown in Figure 10.4 for more than 4 processors. Nevertheless, the costs to coordinate all processors increased which caused a decrease in the speed-up for 32 processors. The achieved speed-up with respect to repeated squaring is close to the predicted one, see column 4 of Table A.24. We conclude that a polynomial basis representation with arbitrary modulus for $\mathbb{F}_{2^n}$ is no suitable data structure for exponentiation. This is true both for sequential and parallel

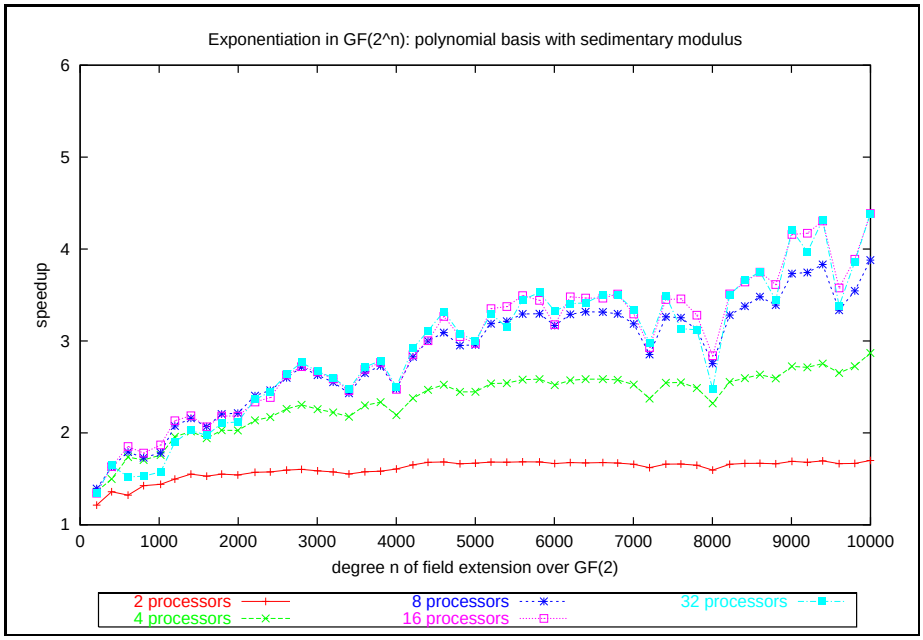


Figure 10.6: The same experiments as illustrated in Figure 10.5 but for test series *Sedimentary*.

computing. The reason is that squaring is expensive both relative to multiplication and in comparison with sparse modulus. This causes poor performance and poor speed-up compared to the other discussed data structures.

The experiments validated the better qualification of a polynomial basis representation with sparse modulus for $\mathbb{F}_{2^n}$ in an impressive way. For 2 processors we had an efficiency[66] of more than 0.8 for all $n \geq 2212$. And even for 4 processors the experiment showed an efficiency of at least 0.6 for most $n \geq 4401$. Figure 10.5 illustrates that the possible speed-up depends on the ratio $c = \frac{c_Q}{c_A}$. For trinomials (only one index in column 2 of Table A.21) the speed-up is higher than for pentanomials since the ratio $c = \frac{c_Q}{c_A}$ (column 3 of Table A.21) is smaller in this case. The faster squaring is significant for parallel exponentiation.

We also observed that the scalability depends on $c = \frac{c_Q}{c_A}$. For $n \leq 4211$ the ratio c is at least 0.07. Algorithm 4.12 runs on between 8 and 16 processors. The experiments showed no great difference between the speed-up for 8, 16 and 32 processors for small n. For $n \geq 4401$ the ratio is at most $c \leq 0.05$ and the number of processors used for Algorithm 4.12 is clearly greater than 16. The experiments showed that the increase of processors from 8 to 16 or even 32 led to clearly higher speed-up.

The speed-up of test series *Sparse* relative to the (sequential) repeated squaring algorithm validated the theoretical results. The scalable algorithm showed speed-ups close to the expected ones for this test series. Thus, the predicted

[66] The efficiency is the quotient of the speed-up divided by the number of used processors.

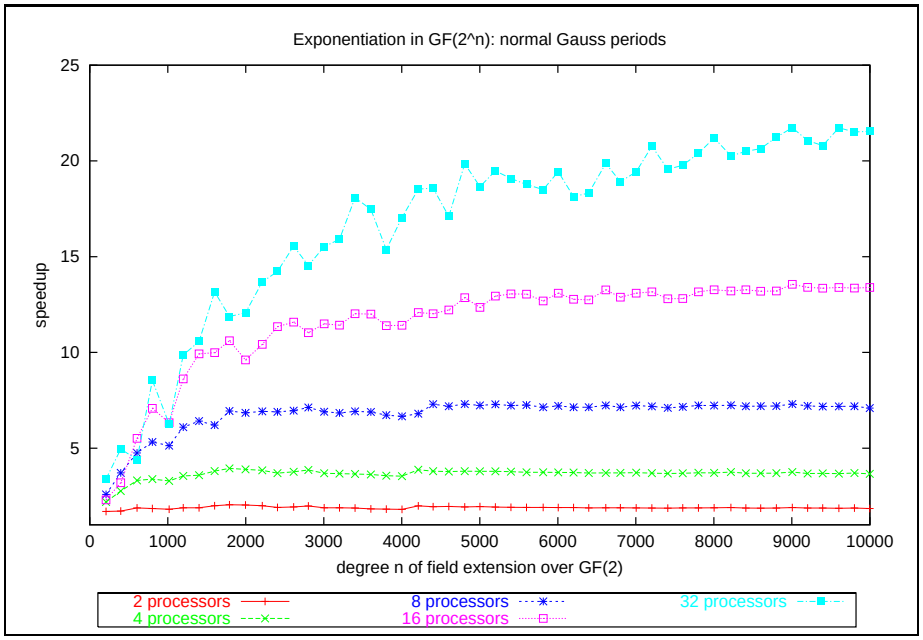


Figure 10.7: The same experiments as illustrated in Figure 10.5 but for the test series *PrimeGP*.

speed-up can be achieved with a moderate number of processors.

We conclude that a sparse modulus is a good choice if the data structure for $\mathbb{F}_{2^n}$ is based on the polynomial basis representation. The sequential arithmetic is much faster than for arbitrary modulus, see Figure 10.9. And exponentiation is well-parallelizable if a moderate number of processors is involved.

The test series *Sedimentary* showed no principal differences in comparison to the test series *Sparse*, as documented in Figure 10.9 and Figure 10.10. In Section 5.2.3, we observed that the slightly slower computation of the canonical representative caused slightly higher values for the ratio c, see also column 2 of Table A.22. Therefore, there were smaller speed-up than for test series *Sparse*. The same scenario has been drawn in Section 4.3 using only theoretical results.

As expected, a normal basis representation showed the best speed-up, see Figure 10.7. The speed-ups of the test series *PrimeGP* are significantly higher than for the test series *Sparse*. For 2 processors the efficiency is more than 0.9 for all values of n but 209 and 398. The efficiency stuck to more than 0.9 for all $n \geq 1601$ using 4 processors, and it was still at least 0.85 for all but two values of n greater than 1601 on a cluster with 8 processor. Thus, exponentiation is very well scalable for a normal basis representation of $\mathbb{F}_{2^n}$.

Fast polynomial multiplication in a normal basis representation transformed this advantage into faster exponentiation, as illustrated in Figure 10.10. Normal prime Gauß periods of type $(n, 1)$ over $\mathbb{F}_2$ already beat a polynomial basis with sparse modulus in sequential, see Figure 10.9. The slightly slower multiplication

Figure 10.8: The maximal *achieved speed-ups* for the for test series *Arbitrary* ($+$), *Sparse* ($\times$), *Sedimentary* ($*$), and *PrimeGP* ($\square$) in relation to the repeated squaring algorithm. The figure is based on experimental data in contrast to Figure 4.10 where the illustration of the *expected speed-ups* is due to theoretical estimates.

(see Table A.14, column 2 and Table A.10, column 2) is compensated by free squarings. For Gauß periods of type $(n, 2)$ over $\mathbb{F}_2$ this compensation is not sufficient to outperform the fastest arithmetic for $\mathbb{F}_2[x]/(f)$, i.e. the modulus f is an irreducible sparse polynomial. Figure 10.9 shows the gap that gains between normal prime Gauß periods of type $(n, 2)$ and polynomial representation with sparse modulus. The situation changes when we compute a power in parallel. Then the higher speed-up comes into play. The breakpoint is between 8 and 16 processors. Figure 10.10 gives the times for parallel exponentiation if 16 processors are involved. For at least this number of processors both types of prime Gauß periods were superior for exponentiation in $\mathbb{F}_{2^n}$. The margin was roughly 2.8 for the Gauß period of type $(9802, 1)$, and 1.01 for type $(9998, 2)$ over $\mathbb{F}_2$.

Conclusions. If normal prime Gauß periods with small parameter $k \in \mathbb{N}_{\geq 1}$ are available then this data structure is best to do parallel exponentiation in $\mathbb{F}_{2^n}$ on massive-parallel machines, i.e. on machines with a large number of processors. Moreover, our experiments in Section 8.1 and Section 9.2 have shown that general Gauß periods often improve the situation if the parameter k is large for prime Gauß periods. Thus, we can draw the following conclusion for the compared basis

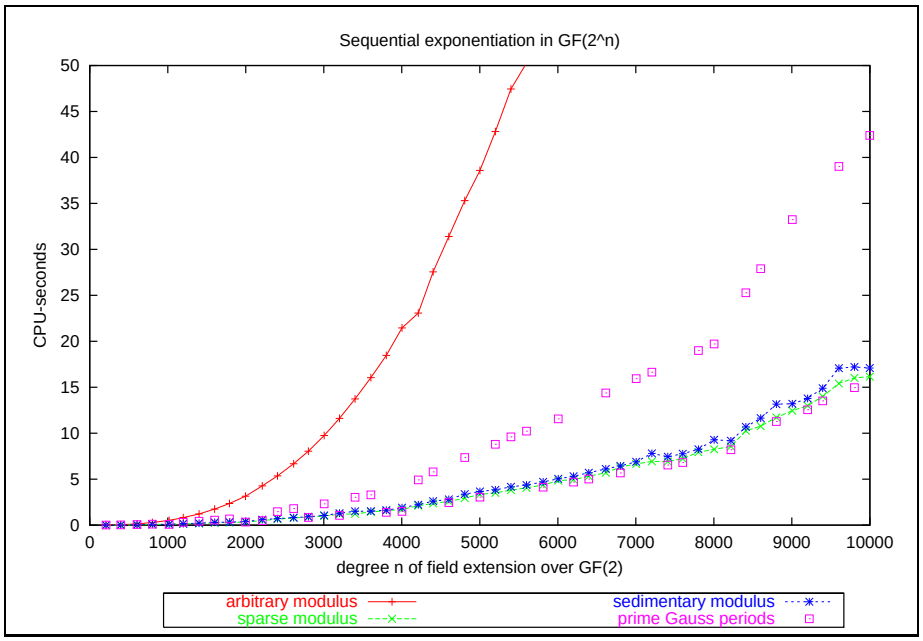


Figure 10.9: Times for sequential exponentiation for the four test series *Arbitrary* (+), *Sparse* (×), *Sedimentary* (∗), and *PrimeGP* (□). The exponentiation algorithm is with respect to a 2-Brauer addition chain.

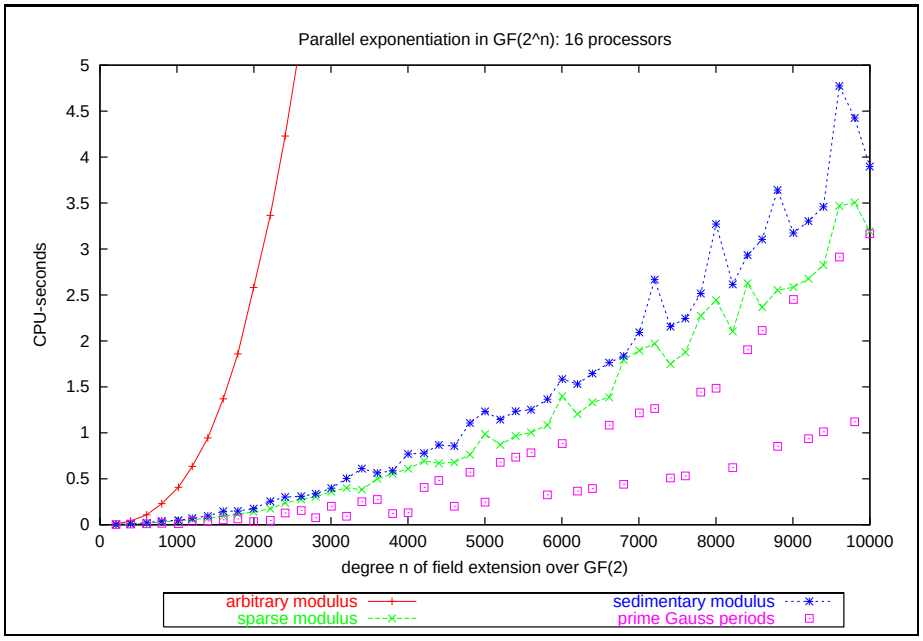


Figure 10.10: Times for parallel exponentiation for the four test series *Arbitrary* (+), *Sparse* (×), *Sedimentary* (∗), and *PrimeGP* (□) on 16 processors.

representations of the finite field $\mathbb{F}_{q^n}$: We claim with respect to our experiments which are in conformity with the theoretical results that general normal Gauß periods offer the best data structure for finite fields on massive-parallel machines.

11. Conclusions

This thesis is concerned with fast exponentiation in finite fields. It was motivated by arithmetic that is used in modern cryptosystems. Our approach to speed up exponentiation is by parallel computing. We have focussed on two main topics to enhance fast parallel exponentiation. The first one discusses bounds on weighted q-addition chains to get a principal understanding on the restrictions of exponentiation with respect to parallel computing. The second topic emphasizes the rôle of different bases representations. Our goal is to profit from fast multiplication in finite fields. In particular, we look at the use of general Gauß periods that define a normal basis.

We first recall our results on the background of weighted q-addition chains. This generalization of addition chains is particularly useful to model exponentiation in finite fields. Although we have only applied this model to this specific task, it is of more general interest. For example, the multiplication of a point on an elliptic curve by a scalar can also be adapted to this model. Thus, the results on exponentiation can be easily transferred to elliptic curve cryptosystems.

We have found a lower bound on parallel exponentiation for q-addition chains with weight (c_Q, c_A). This lower bound is not strict for all weights, but it coincides with previously known lower bounds in two important special cases. Therefore, our general bound closes the gap between original addition chains with weight $(1, 1)$ and addition chains with free scalar, i.e. weight $(0, 1)$. As shown by experiment, most polynomial basis representations for finite fields have arithmetic properties that are related to this gap. This gap is characterized by the ratio $c = \frac{c_Q}{c_A}$ of the cost between a q-step, i.e. an evaluation of the Frobenius automorphism in $\mathbb{F}_{q^n}$ with c_Q operations in $\mathbb{F}_q$, and a non-q-step, i.e. a multiplication in $\mathbb{F}_{q^n}$ with cost c_A.

The ratio c can be seen as the indicator whether exponentiation can be parallelized successfully. We have used this observation to develop a new parallel algorithm for exponentiation. Starting from the optimal algorithm of Borodin & Munro (1975) for original addition chains, we have taken this ratio into account. It does not only determine the depth of the generalized algorithm but also the necessary number of processors. In the special case $c = 0$, the algorithm was formerly described by von zur Gathen (1991). The ratio $c = 0$ describes the arithmetic properties of a normal basis representation of $\mathbb{F}_{q^n}$. As for the lower bound, our general approach connects both previously known algorithms. Our results support the assumption that the lower bound is close to the depth in all discussed cases.

As a second point, we have considered fast arithmetic for normal bases. We present a parallel algorithm for inversion in $\mathbb{F}_{q^n}$. It has optimal depth in case $\mathbb{F}_{2^n}$ where it uses only 2 processors by taking advantage of the special form of the exponent e used for inversion. This improves the general algorithm with respect to the number of processors in an important case.

Our main result on normal bases concerns normal general Gauß periods. For these particular normal elements, we overcome the slow matrix-based multiplication routine by fast polynomial multiplication. Therefore, we have translated the relation between Gauß periods and cyclotomic polynomials into an algorithmic benefit. This approach was already described for prime Gauß periods by Gao *et al.* (2000). We have generalized their algorithm to prime power Gauß periods to make fast polynomial multiplication and inversion available.

Decomposable Gauß periods have been our tool to extend this result to general Gauß periods. With their help, we prove that fast polynomial multiplication can be used in a normal basis representation of $\mathbb{F}_{q^n}$ whenever there is a normal Gauß period of type $(n, \mathcal{K})$ over $\mathbb{F}_q$.

In the closing section, we discuss some problems that arise when exponentiation in finite fields is implemented on real parallel machines. We have developed a scalable parallel exponentiation algorithm with respect to our results that takes the limited number of processors into account. The algorithm also tries to minimize communication costs. All key algorithms of this thesis have been implemented and compared by experiments. These experiments support our theoretical results. They show the advantages (and disadvantages) of the discussed basis representations for $\mathbb{F}_{2^n}$ with respect to fast (parallel) exponentiation in detail. In particular, they support the choice of normal general Gauß periods of type $(n, \mathcal{K})$ over $\mathbb{F}_q$ as the preferred data structure for parallel exponentiation whenever $\mathcal{K}$ has small order. Therefore, we are optimistic that the results of this thesis are of use to cryptography and other applications of finite fields where fast arithmetic, in particular fast exponentiation, plays an important rôle.

A. Appendix: Experiments

A.1. Addition chains. Tables A.1–A.5 document in detail the results of the experiment that is described in Section 3.4. The entries of these tables are the input for Figures 3.1–3.3. All sets of weights consist of 50 different values $n \approx 200i$, $1 \leq i \leq 50$. Each row includes the average values for 10000 trials on input n and c.

The key to Tables A.1–A.5		
column	label	description
1–2	Param.	Input parameters of the 2-addition chain with weight (c_Q, c_A) for an exponent $e \in \mathbb{N}_{\geq 1}$.
1	n	The binary length of the exponent e.
2	c	The ratio $c = \frac{c_Q}{c_A}$ of the weight (c_Q, c_A).
3–5	binary	The average number of steps of the binary addition chain (p. 21) on 10000 randomly chosen exponents e of binary length n.
3	A	The number of non-doubling steps.
4	Q	The number of doublings.
5	L	The normalized weighted length $L = A + c \cdot Q$ of the binary addition chain.
6–10	Brauer	The average number of steps and some parameters of the Brauer 2-addition chain (Algorithm 3.12) on the same 10000 randomly chosen exponents as for the binary addition chain.
6	m	The tuning parameter; we have chosen the value by hand such that L (column 9) is minimal.
7–9	A, Q, L	See columns 3–6.
10	spd.	The ratio of the entries of column 5 to those of column 9. This illustrates the improvement of a Brauer 2-addition chain compared to the binary addition chain.
11–15	BGMW	The average number of steps and some parameters of the BGMW 2-addition chain (Algorithm 3.25).
11–14	m, A, Q, L	See columns 6–10.
15	spd.	The ratio of the entries of column 5 to those of column 14.

Addition chains with weighted length: Kung's model														
Param.		binary			Brauer					BGMW				
n	c	A	Q	L	m	A	Q	L	spd.	m	A	Q	L	spd.
209	1.00	103	208	311	4	55	212	267	1.16	4	62	212	274	1.14
398	1.00	198	397	595	5	91	408	499	1.19	4	106	400	506	1.18
606	1.00	302	605	907	5	131	616	747	1.21	5	147	610	757	1.20
803	1.00	400	802	1202	5	169	813	982	1.22	5	184	805	989	1.22
1018	1.00	508	1017	1525	5	211	1028	1239	1.23	5	226	1020	1246	1.22
1199	1.00	599	1198	1797	6	226	1224	1450	1.24	6	257	1200	1457	1.23
1401	1.00	700	1400	2100	6	260	1426	1686	1.25	6	291	1404	1695	1.24
1601	1.00	800	1600	2400	6	292	1626	1918	1.25	6	323	1602	1925	1.25
1791	1.00	894	1790	2684	6	324	1816	2140	1.25	6	355	1794	2149	1.25
1996	1.00	997	1995	2992	6	357	2021	2378	1.26	6	388	1998	2386	1.25
2212	1.00	1106	2211	3317	6	393	2237	2630	1.26	6	424	2214	2638	1.26
2406	1.00	1202	2405	3607	6	424	2431	2855	1.26	6	455	2406	2861	1.26
2613	1.00	1306	2612	3918	6	459	2638	3097	1.27	6	490	2616	3106	1.26
2802	1.00	1400	2801	4201	6	489	2827	3316	1.27	6	520	2802	3322	1.26
3005	1.00	1501	3004	4505	7	488	3061	3549	1.27	6	554	3006	3560	1.27
3202	1.00	1600	3201	4801	7	516	3258	3774	1.27	7	579	3206	3785	1.27
3401	1.00	1700	3400	5100	7	544	3457	4001	1.27	7	607	3402	4009	1.27
3603	1.00	1801	3602	5403	7	572	3659	4231	1.28	7	635	3605	4240	1.27
3802	1.00	1900	3801	5701	7	601	3858	4459	1.28	7	664	3808	4472	1.27
4002	1.00	2000	4001	6001	7	629	4058	4687	1.28	7	692	4004	4696	1.28
4211	1.00	2105	4210	6315	7	659	4267	4926	1.28	7	722	4214	4936	1.28
4401	1.00	2200	4400	6600	7	686	4457	5143	1.28	7	749	4403	5152	1.28
4602	1.00	2300	4601	6901	7	714	4658	5372	1.28	7	777	4606	5383	1.28
4806	1.00	2402	4805	7207	7	743	4862	5605	1.29	7	806	4809	5615	1.28
5002	1.00	2500	5001	7501	7	771	5058	5829	1.29	7	834	5005	5839	1.28
5199	1.00	2599	5198	7797	7	799	5255	6054	1.29	7	862	5201	6063	1.29
5399	1.00	2698	5398	8096	7	827	5455	6282	1.29	7	891	5404	6295	1.29
5598	1.00	2798	5597	8395	7	855	5654	6509	1.29	7	918	5600	6518	1.29
5812	1.00	2905	5811	8716	7	886	5868	6754	1.29	7	949	5817	6766	1.29
6005	1.00	3002	6004	9006	7	913	6061	6974	1.29	7	976	6006	6982	1.29
6202	1.00	3100	6201	9301	7	941	6258	7199	1.29	7	1004	6202	7206	1.29
6396	1.00	3197	6395	9592	7	968	6452	7420	1.29	7	1031	6398	7429	1.29
6614	1.00	3306	6613	9919	7	999	6670	7669	1.29	7	1062	6615	7677	1.29
6802	1.00	3400	6801	10201	7	1026	6858	7884	1.29	7	1089	6804	7893	1.29
7005	1.00	3501	7004	10505	7	1055	7061	8116	1.29	7	1118	7007	8125	1.29
7205	1.00	3600	7204	10804	7	1083	7261	8344	1.29	7	1146	7210	8356	1.29
7410	1.00	3704	7409	11113	7	1112	7466	8578	1.30	7	1175	7413	8588	1.29
7602	1.00	3801	7601	11402	8	1073	7721	8794	1.30	7	1202	7602	8804	1.30
7803	1.00	3900	7802	11702	8	1098	7922	9020	1.30	8	1225	7808	9033	1.30
8003	1.00	4000	8002	12002	8	1122	8122	9244	1.30	8	1250	8008	9258	1.30
8218	1.00	4108	8217	12325	8	1149	8337	9486	1.30	8	1276	8224	9500	1.30
8411	1.00	4205	8410	12615	8	1173	8530	9703	1.30	8	1300	8416	9716	1.30
8601	1.00	4299	8600	12899	8	1197	8720	9917	1.30	8	1324	8608	9932	1.30
8802	1.00	4400	8801	13201	8	1222	8921	10143	1.30	8	1349	8808	10157	1.30
9006	1.00	4502	9005	13507	8	1247	9125	10372	1.30	8	1374	9008	10382	1.30
9202	1.00	4599	9201	13800	8	1272	9321	10593	1.30	8	1399	9208	10607	1.30
9396	1.00	4697	9395	14092	8	1296	9515	10811	1.30	8	1423	9400	10823	1.30
9603	1.00	4801	9602	14403	8	1322	9722	11044	1.30	8	1449	9608	11057	1.30
9802	1.00	4900	9801	14701	8	1346	9921	11267	1.30	8	1474	9808	11282	1.30
9998	1.00	4999	9997	14996	8	1371	10117	11488	1.31	8	1498	10000	11498	1.30

Table A.1: Comparison of different 2-addition chain algorithms if doublings and non-doublings have same weight. This is assumed for Kung's model.

Addition chains with weighted length: arbitrary modulus														
Param.		binary			Brauer					BGMW				
n	c	A	Q	L	m	A	Q	L	spd.	m	A	Q	L	spd.
209	0.74	103	208	257	4	55	212	212	1.21	4	62	212	219	1.17
398	0.64	198	397	453	5	91	408	353	1.28	4	106	400	363	1.25
606	0.61	302	605	669	5	131	616	504	1.33	5	147	610	517	1.29
803	0.64	400	802	913	6	161	828	691	1.32	5	184	805	699	1.31
1018	0.69	508	1017	1213	6	197	1043	920	1.32	5	226	1020	933	1.30
1199	0.62	599	1198	1340	6	226	1224	984	1.36	6	257	1200	1000	1.34
1401	0.63	700	1400	1578	6	260	1426	1155	1.37	6	291	1404	1172	1.35
1601	0.64	800	1600	1830	6	292	1626	1339	1.37	6	323	1602	1354	1.35
1791	0.67	894	1790	2089	6	324	1816	1536	1.36	6	355	1794	1553	1.35
1996	0.70	997	1995	2390	6	357	2021	1768	1.35	6	388	1998	1783	1.34
2212	0.64	1106	2211	2529	6	393	2237	1833	1.38	6	424	2214	1849	1.37
2406	0.64	1202	2405	2746	7	403	2462	1983	1.38	6	455	2406	1999	1.37
2613	0.65	1306	2612	2996	7	432	2669	2159	1.39	6	490	2616	2183	1.37
2802	0.65	1400	2801	3234	7	459	2858	2330	1.39	6	520	2802	2354	1.37
3005	0.67	1501	3004	3513	7	488	3061	2538	1.38	6	554	3006	2567	1.37
3202	0.67	1600	3201	3742	7	516	3258	2696	1.39	7	579	3206	2725	1.37
3401	0.68	1700	3400	4014	7	544	3457	2897	1.39	7	607	3402	2922	1.37
3603	0.70	1801	3602	4311	7	572	3659	3122	1.38	7	635	3605	3147	1.37
3802	0.71	1900	3801	4596	7	601	3858	3337	1.38	7	664	3808	3365	1.37
4002	0.73	2000	4001	4904	7	629	4058	3574	1.37	7	692	4004	3598	1.36
4211	0.68	2105	4210	4950	7	659	4267	3542	1.40	7	722	4214	3569	1.39
4401	0.67	2200	4400	5157	7	686	4457	3682	1.40	7	749	4403	3708	1.39
4602	0.68	2300	4601	5405	7	714	4658	3858	1.40	7	777	4606	3886	1.39
4806	0.67	2402	4805	5609	7	743	4862	3988	1.41	7	806	4809	4015	1.40
5002	0.67	2500	5001	5829	7	771	5058	4138	1.41	7	834	5005	4165	1.40
5199	0.66	2599	5198	6028	7	799	5255	4266	1.41	7	862	5201	4293	1.40
5399	0.67	2698	5398	6310	7	827	5455	4477	1.41	7	891	5404	4507	1.40
5598	0.66	2798	5597	6483	7	855	5654	4578	1.42	7	918	5600	4605	1.41
5812	0.67	2905	5811	6783	7	886	5868	4802	1.41	7	949	5817	4831	1.40
6005	0.67	3002	6004	7039	7	913	6061	4989	1.41	7	976	6006	5015	1.40
6202	0.65	3100	6201	7153	7	941	6258	5032	1.42	7	1004	6202	5058	1.41
6396	0.67	3197	6395	7495	7	968	6452	5304	1.41	7	1031	6398	5331	1.41
6614	0.67	3306	6613	7729	8	949	6733	5452	1.42	7	1062	6615	5487	1.41
6802	0.64	3400	6801	7734	8	973	6921	5384	1.44	7	1089	6804	5425	1.43
7005	0.67	3501	7004	8167	8	998	7124	5744	1.42	7	1118	7007	5786	1.41
7205	0.67	3600	7204	8425	8	1023	7324	5929	1.42	7	1146	7210	5975	1.41
7410	0.67	3704	7409	8681	8	1049	7529	6107	1.42	7	1175	7413	6155	1.41
7602	0.67	3801	7601	8889	8	1073	7721	6242	1.42	8	1200	7608	6293	1.41
7803	0.67	3900	7802	9120	8	1098	7922	6398	1.43	8	1225	7808	6449	1.41
8003	0.67	4000	8002	9341	8	1122	8122	6544	1.43	8	1250	8008	6595	1.42
8218	0.66	4108	8217	9564	8	1149	8337	6685	1.43	8	1276	8224	6737	1.42
8411	0.66	4205	8410	9747	8	1173	8530	6794	1.43	8	1300	8416	6846	1.42
8601	0.66	4299	8600	9957	8	1197	8720	6933	1.44	8	1324	8608	6987	1.43
8802	0.67	4400	8801	10258	8	1222	8921	7160	1.43	8	1349	8808	7211	1.42
9006	0.66	4502	9005	10489	8	1247	9125	7314	1.43	8	1374	9008	7363	1.42
9202	0.67	4599	9201	10761	8	1272	9321	7515	1.43	8	1399	9208	7566	1.42
9396	0.66	4697	9395	10873	8	1296	9515	7551	1.44	8	1423	9400	7603	1.43
9603	0.65	4801	9602	11077	8	1322	9722	7676	1.44	8	1449	9608	7729	1.43
9802	0.66	4900	9801	11395	8	1346	9921	7920	1.44	8	1474	9808	7973	1.43
9998	0.66	4999	9997	11548	8	1371	10117	7999	1.44	8	1498	10000	8049	1.43

Table A.2: Comparison of different 2-addition chain algorithms if doublings and non-doublings have quotient roughly 2/3. This is the case if division is computed via Newton inversion as described in Section 5.2.1.

Addition chains with weighted length: sparse modulus														
Param.		binary			Brauer					BGMW				
n	c	A	Q	L	m	A	Q	L	spd.	m	A	Q	L	spd.
209	0.48	103	208	202	5	54	219	158	1.28	4	62	212	163	1.24
398	0.29	198	397	313	5	91	408	209	1.49	4	106	400	222	1.41
606	0.18	302	605	410	6	129	631	241	1.70	5	147	610	255	1.60
803	0.18	400	802	546	6	161	828	311	1.75	5	184	805	330	1.65
1018	0.19	508	1017	704	6	197	1043	398	1.77	5	226	1020	423	1.67
1199	0.11	599	1198	734	6	226	1224	364	2.02	6	257	1200	392	1.87
1401	0.10	700	1400	846	6	260	1426	409	2.07	6	291	1404	438	1.93
1601	0.10	800	1600	959	6	292	1626	454	2.11	6	323	1602	483	1.99
1791	0.10	894	1790	1077	7	315	1847	504	2.14	6	355	1794	538	2.00
1996	0.10	997	1995	1205	7	345	2052	559	2.15	6	388	1998	596	2.02
2212	0.08	1106	2211	1277	7	375	2268	551	2.32	6	424	2214	595	2.14
2406	0.08	1202	2405	1404	7	403	2462	609	2.30	6	455	2406	657	2.14
2613	0.08	1306	2612	1517	7	432	2669	647	2.34	6	490	2616	701	2.16
2802	0.08	1400	2801	1618	7	459	2858	682	2.37	6	520	2802	739	2.19
3005	0.08	1501	3004	1739	7	488	3061	731	2.38	7	551	3010	790	2.20
3202	0.08	1600	3201	1842	7	516	3258	762	2.42	7	579	3206	821	2.24
3401	0.06	1700	3400	1918	7	544	3457	766	2.50	7	607	3402	825	2.32
3603	0.08	1801	3602	2077	7	572	3659	852	2.44	7	635	3605	911	2.28
3802	0.08	1900	3801	2189	7	601	3858	895	2.45	7	664	3808	954	2.29
4002	0.08	2000	4001	2315	7	629	4058	948	2.44	7	692	4004	1007	2.30
4211	0.07	2105	4210	2382	7	659	4267	940	2.53	7	722	4214	999	2.38
4401	0.05	2200	4400	2419	7	686	4457	908	2.66	7	749	4403	968	2.50
4602	0.05	2300	4601	2523	8	699	4721	928	2.72	7	777	4606	1001	2.52
4806	0.05	2402	4805	2622	8	724	4925	950	2.76	7	806	4809	1026	2.55
5002	0.05	2500	5001	2770	8	749	5121	1025	2.70	7	834	5005	1104	2.51
5199	0.04	2599	5198	2827	8	773	5318	1006	2.81	7	862	5201	1090	2.59
5399	0.04	2698	5398	2929	8	798	5518	1035	2.83	7	891	5404	1123	2.61
5598	0.04	2798	5597	3038	8	823	5717	1068	2.84	7	918	5600	1158	2.62
5812	0.04	2905	5811	3152	8	850	5931	1102	2.86	7	949	5817	1196	2.63
6005	0.05	3002	6004	3310	8	874	6124	1189	2.78	7	976	6006	1284	2.58
6202	0.04	3100	6201	3363	8	898	6321	1166	2.88	7	1004	6202	1267	2.65
6396	0.04	3197	6395	3463	8	922	6515	1193	2.90	7	1031	6398	1297	2.67
6614	0.04	3306	6613	3580	8	949	6733	1228	2.91	7	1062	6615	1336	2.68
6802	0.05	3400	6801	3733	8	973	6921	1312	2.85	7	1089	6804	1422	2.62
7005	0.05	3501	7004	3850	8	998	7124	1353	2.84	7	1118	7007	1467	2.62
7205	0.05	3600	7204	3951	8	1023	7324	1379	2.86	7	1146	7210	1497	2.64
7410	0.04	3704	7409	4012	8	1049	7529	1362	2.94	7	1175	7413	1484	2.70
7602	0.04	3801	7601	4123	8	1073	7721	1400	2.94	8	1200	7608	1523	2.71
7803	0.05	3900	7802	4300	8	1098	7922	1504	2.86	8	1225	7808	1625	2.65
8003	0.05	4000	8002	4420	8	1122	8122	1548	2.85	8	1250	8008	1670	2.65
8218	0.04	4108	8217	4448	8	1149	8337	1494	2.98	8	1276	8224	1617	2.75
8411	0.04	4205	8410	4565	8	1173	8530	1538	2.97	8	1300	8416	1660	2.75
8601	0.03	4299	8600	4595	8	1197	8720	1497	3.07	8	1324	8608	1620	2.84
8802	0.03	4400	8801	4687	8	1222	8921	1513	3.10	8	1349	8808	1636	2.86
9006	0.03	4502	9005	4788	8	1247	9125	1537	3.11	8	1374	9008	1661	2.88
9202	0.03	4599	9201	4890	8	1272	9321	1567	3.12	8	1399	9208	1690	2.89
9396	0.03	4697	9395	4980	8	1296	9515	1583	3.15	8	1423	9400	1706	2.92
9603	0.04	4801	9602	5145	8	1322	9722	1670	3.08	8	1449	9608	1793	2.87
9802	0.03	4900	9801	5241	8	1346	9921	1691	3.10	8	1474	9808	1815	2.89
9998	0.03	4999	9997	5296	8	1371	10117	1671	3.17	8	1498	10000	1795	2.95

Table A.3: Comparison of different 2-addition chain algorithms if doublings and non-doublings decrease. The values for c are valid for the case when the modulus is a sparse polynomial as described in Section 5.2.2.

Addition chains with weighted length: sedimentary modulus														
Param.		binary			Brauer					BGMW				
n	c	A	Q	L	m	A	Q	L	spd.	m	A	Q	L	spd.
209	0.57	103	208	221	5	54	219	179	1.24	4	62	212	183	1.21
398	0.32	198	397	325	5	91	408	222	1.47	4	106	400	234	1.39
606	0.24	302	605	448	6	129	631	281	1.59	5	147	610	294	1.52
803	0.24	400	802	590	6	161	828	358	1.65	5	184	805	375	1.57
1018	0.21	508	1017	718	6	197	1043	412	1.74	5	226	1020	436	1.64
1199	0.15	599	1198	781	6	226	1224	412	1.89	6	257	1200	439	1.78
1401	0.14	700	1400	894	6	260	1426	457	1.95	6	291	1404	485	1.84
1601	0.16	800	1600	1049	6	292	1626	545	1.92	6	323	1602	572	1.83
1791	0.13	894	1790	1133	7	315	1847	562	2.02	6	355	1794	595	1.90
1996	0.13	997	1995	1257	7	345	2052	613	2.05	6	388	1998	649	1.94
2212	0.11	1106	2211	1352	7	375	2268	627	2.15	6	424	2214	670	2.02
2406	0.10	1202	2405	1452	7	403	2462	658	2.20	6	455	2406	705	2.06
2613	0.09	1306	2612	1544	7	432	2669	675	2.29	6	490	2616	728	2.12
2802	0.08	1400	2801	1634	7	459	2858	698	2.34	6	520	2802	754	2.17
3005	0.09	1501	3004	1769	7	488	3061	761	2.32	6	554	3006	822	2.15
3202	0.09	1600	3201	1898	7	516	3258	819	2.32	7	579	3206	877	2.16
3401	0.10	1700	3400	2046	7	544	3457	896	2.28	7	607	3402	953	2.15
3603	0.09	1801	3602	2110	7	572	3659	886	2.38	7	635	3605	945	2.23
3802	0.08	1900	3801	2204	7	601	3858	909	2.42	7	664	3808	968	2.28
4002	0.10	2000	4001	2388	7	629	4058	1022	2.34	7	692	4004	1080	2.21
4211	0.07	2105	4210	2419	7	659	4267	977	2.47	7	722	4214	1036	2.33
4401	0.07	2200	4400	2493	7	686	4457	983	2.54	7	749	4403	1043	2.39
4602	0.06	2300	4601	2583	8	699	4721	989	2.61	7	777	4606	1060	2.44
4806	0.07	2402	4805	2729	8	724	4925	1059	2.58	7	806	4809	1133	2.41
5002	0.07	2500	5001	2839	8	749	5121	1096	2.59	7	834	5005	1173	2.42
5199	0.06	2599	5198	2905	8	773	5318	1086	2.67	7	862	5201	1168	2.49
5399	0.06	2698	5398	3007	8	798	5518	1114	2.70	7	891	5404	1201	2.50
5598	0.05	2798	5597	3100	8	823	5717	1132	2.74	7	918	5600	1221	2.54
5812	0.05	2905	5811	3217	8	850	5931	1168	2.75	7	949	5817	1261	2.55
6005	0.06	3002	6004	3350	8	874	6124	1229	2.73	7	976	6006	1324	2.53
6202	0.05	3100	6201	3431	8	898	6321	1235	2.78	7	1004	6202	1335	2.57
6396	0.05	3197	6395	3531	8	922	6515	1262	2.80	7	1031	6398	1365	2.59
6614	0.05	3306	6613	3651	8	949	6733	1300	2.81	7	1062	6615	1407	2.59
6802	0.05	3400	6801	3754	8	973	6921	1334	2.81	7	1089	6804	1444	2.60
7005	0.06	3501	7004	3891	8	998	7124	1394	2.79	7	1118	7007	1508	2.58
7205	0.07	3600	7204	4098	8	1023	7324	1529	2.68	7	1146	7210	1644	2.49
7410	0.05	3704	7409	4096	8	1049	7529	1448	2.83	7	1175	7413	1567	2.61
7602	0.05	3801	7601	4209	8	1073	7721	1487	2.83	8	1200	7608	1608	2.62
7803	0.06	3900	7802	4349	8	1098	7922	1554	2.80	8	1225	7808	1674	2.60
8003	0.07	4000	8002	4575	8	1122	8122	1705	2.68	8	1250	8008	1825	2.51
8218	0.05	4108	8217	4533	8	1149	8337	1580	2.87	8	1276	8224	1701	2.66
8411	0.05	4205	8410	4620	8	1173	8530	1594	2.90	8	1300	8416	1715	2.69
8601	0.05	4299	8600	4695	8	1197	8720	1599	2.94	8	1324	8608	1721	2.73
8802	0.05	4400	8801	4829	8	1222	8921	1657	2.91	8	1349	8808	1778	2.71
9006	0.04	4502	9005	4861	8	1247	9125	1610	3.02	8	1374	9008	1733	2.80
9202	0.04	4599	9201	4964	8	1272	9321	1641	3.02	8	1399	9208	1764	2.81
9396	0.04	4697	9395	5056	8	1296	9515	1660	3.05	8	1423	9400	1782	2.84
9603	0.05	4801	9602	5277	8	1322	9722	1804	2.92	8	1449	9608	1926	2.74
9802	0.04	4900	9801	5335	8	1346	9921	1786	2.99	8	1474	9808	1909	2.79
9998	0.04	4999	9997	5372	8	1371	10117	1749	3.07	8	1498	10000	1872	2.87

Table A.4: Comparison of different 2-addition chain algorithms if doublings and non-doublings decrease. The values for c are valid for the case when the modulus is a sedimentary polynomial as described in Section 5.2.3.

Addition chains with weighted length: free doublings														
Param.		binary			Brauer					BGMW				
n	c	A	Q	L	m	A	Q	L	spd.	m	A	Q	L	spd.
209	0.00	103	208	103	4	55	212	55	1.87	3	66	210	66	1.56
398	0.00	198	397	198	5	91	408	91	2.18	4	106	400	106	1.87
606	0.00	302	605	302	6	129	631	129	2.34	5	147	610	147	2.05
803	0.00	400	802	400	6	161	828	161	2.48	5	184	805	184	2.17
1018	0.00	508	1017	508	6	197	1043	197	2.58	5	226	1020	226	2.25
1199	0.00	599	1198	599	6	226	1224	226	2.65	6	257	1200	257	2.33
1401	0.00	700	1400	700	6	260	1426	260	2.69	6	291	1404	291	2.41
1601	0.00	800	1600	800	7	289	1657	289	2.77	6	323	1602	323	2.48
1791	0.00	894	1790	894	7	315	1847	315	2.84	6	355	1794	355	2.52
1996	0.00	997	1995	997	7	345	2052	345	2.89	6	388	1998	388	2.57
2212	0.00	1106	2211	1106	7	375	2268	375	2.95	6	424	2214	424	2.61
2406	0.00	1202	2405	1202	7	403	2462	403	2.98	6	455	2406	455	2.64
2613	0.00	1306	2612	1306	7	432	2669	432	3.02	6	490	2616	490	2.67
2802	0.00	1400	2801	1400	7	459	2858	459	3.05	6	520	2802	520	2.69
3005	0.00	1501	3004	1501	7	488	3061	488	3.08	7	551	3010	551	2.72
3202	0.00	1600	3201	1600	7	516	3258	516	3.10	7	579	3206	579	2.76
3401	0.00	1700	3400	1700	7	544	3457	544	3.12	7	607	3402	607	2.80
3603	0.00	1801	3602	1801	7	572	3659	572	3.15	7	635	3605	635	2.84
3802	0.00	1900	3801	1900	8	599	3921	599	3.17	7	664	3808	664	2.86
4002	0.00	2000	4001	2000	8	624	4121	624	3.21	7	692	4004	692	2.89
4211	0.00	2105	4210	2105	8	650	4330	650	3.24	7	722	4214	722	2.92
4401	0.00	2200	4400	2200	8	674	4520	674	3.26	7	749	4403	749	2.94
4602	0.00	2300	4601	2300	8	699	4721	699	3.29	7	777	4606	777	2.96
4806	0.00	2402	4805	2402	8	724	4925	724	3.32	7	806	4809	806	2.98
5002	0.00	2500	5001	2500	8	749	5121	749	3.34	7	834	5005	834	3.00
5199	0.00	2599	5198	2599	8	773	5318	773	3.36	7	862	5201	862	3.02
5399	0.00	2698	5398	2698	8	798	5518	798	3.38	7	891	5404	891	3.03
5598	0.00	2798	5597	2798	8	823	5717	823	3.40	7	918	5600	918	3.05
5812	0.00	2905	5811	2905	8	850	5931	850	3.42	7	949	5817	949	3.06
6005	0.00	3002	6004	3002	8	874	6124	874	3.43	7	976	6006	976	3.08
6202	0.00	3100	6201	3100	8	898	6321	898	3.45	7	1004	6202	1004	3.09
6396	0.00	3197	6395	3197	8	922	6515	922	3.47	7	1031	6398	1031	3.10
6614	0.00	3306	6613	3306	8	949	6733	949	3.48	7	1062	6615	1062	3.11
6802	0.00	3400	6801	3400	8	973	6921	973	3.49	7	1089	6804	1089	3.12
7005	0.00	3501	7004	3501	8	998	7124	998	3.51	7	1118	7007	1118	3.13
7205	0.00	3600	7204	3600	8	1023	7324	1023	3.52	7	1146	7210	1146	3.14
7410	0.00	3704	7409	3704	8	1049	7529	1049	3.53	7	1175	7413	1175	3.15
7602	0.00	3801	7601	3801	8	1073	7721	1073	3.54	8	1200	7608	1200	3.17
7803	0.00	3900	7802	3900	8	1098	7922	1098	3.55	8	1225	7808	1225	3.18
8003	0.00	4000	8002	4000	8	1122	8122	1122	3.57	8	1250	8008	1250	3.20
8218	0.00	4108	8217	4108	8	1149	8337	1149	3.58	8	1276	8224	1276	3.22
8411	0.00	4205	8410	4205	8	1173	8530	1173	3.58	8	1300	8416	1300	3.23
8601	0.00	4299	8600	4299	8	1197	8720	1197	3.59	8	1324	8608	1324	3.25
8802	0.00	4400	8801	4400	8	1222	8921	1222	3.60	8	1349	8808	1349	3.26
9006	0.00	4502	9005	4502	8	1247	9125	1247	3.61	8	1374	9008	1374	3.28
9202	0.00	4599	9201	4599	8	1272	9321	1272	3.62	8	1399	9208	1399	3.29
9396	0.00	4697	9395	4697	9	1295	9642	1295	3.63	8	1423	9400	1423	3.30
9603	0.00	4801	9602	4801	9	1318	9849	1318	3.64	8	1449	9608	1449	3.31
9802	0.00	4900	9801	4900	9	1341	10048	1341	3.65	8	1474	9808	1474	3.32
9998	0.00	4999	9997	4999	9	1362	10244	1362	3.67	8	1498	10000	1498	3.34

Table A.5: Comparison of different 2-addition chain algorithms if doublings are free and non-doublings have unit cost. This situation is given for normal basis representations of finite fields.

Tables A.6–A.8 list the calculations that are discussed in Section 4.3. All values are computed due to the theoretical results of Section 4. The different sets of weights are the same as in Tables A.2–A.4.

The key to Tables A.6–A.8		
column	label	description
1–3	Param.	Input parameters for a 2-addition chain with weight (c_Q, c_A) for an exponent $e \in \mathbb{N}_{\geq 1}$.
1	n	The binary length of the exponent e.
2	c	The ratio $c = \frac{c_Q}{c_A}$ of the weight (c_Q, c_A).
3	P	The number of processors to execute Algorithm 4.12 on input e and c.
4	binary	The average normalized length $L = c \cdot (n-1) + \frac{1}{2}(n-1)$ of a binary addition chain for a randomly chosen exponent e of exactly n bits. This length is the actuarial expectation in contrast to column 5 of Tables A.2–A.4 where the average length is given by experiment on 10000 randomly chosen exponents.
5–7	lower bound	The lower bounds on the depth for some exponents due to Result 4.17.
5	$e = 2^{n-1}$	The lower bound for the n-bit exponent 2^{n-1}.
6	$e = 2^n - 1$	The lower bound for the n-bit exponent $2^n - 1$.
7	Δ	The gap between column 5 and column 6.
8–10	Algorithm 4.12	Values that quantify the quality of Algorithm 4.12 in comparison to the lower bound (columns 5–7) and the binary addition chain (column 4).
8	depth	The estimate on the depth for Algorithm 4.12 as given in Theorem 4.13.
9	Δ	The gap between the lower bound for the exponent $e = 2^n - 1$ (column 6) and the depth given in column 7.
10	spd.	The speed-up of Algorithm 4.12 relative to the average length of the binary addition chain. This is computed as quotient of column 4 and column 8.

Addition chains with weighted length: arbitrary modulus									
Param.			binary	lower bound			Algorithm 4.12		
n	c	P	av. L	$e = 2^{n-1}$	$e = 2^n - 1$	Δ	depth	Δ	spd.
209	0.74	3	259	154.80	155.80	1	156.80	1.00	1.65
398	0.64	3	454	255.49	256.49	1	257.49	1.00	1.76
606	0.61	3	670	367.30	368.30	1	369.30	1.00	1.81
803	0.64	3	915	513.64	514.64	1	515.64	1.00	1.77
1018	0.69	3	1214	705.04	706.04	1	707.04	1.00	1.72
1199	0.62	3	1341	742.00	743.00	1	744.00	1.00	1.80
1401	0.63	3	1579	878.82	879.82	1	880.82	1.00	1.79
1601	0.64	3	1831	1030.63	1031.63	1	1032.63	1.00	1.77
1791	0.67	3	2090	1195.42	1196.42	1	1197.42	1.00	1.75
1996	0.70	3	2391	1393.43	1394.43	1	1395.43	1.00	1.71
2212	0.64	3	2529	1423.94	1424.94	1	1425.94	1.00	1.77
2406	0.64	3	2747	1544.34	1545.34	1	1546.34	1.00	1.78
2613	0.65	3	2997	1690.56	1691.56	1	1692.56	1.00	1.77
2802	0.65	3	3235	1834.13	1835.13	1	1836.13	1.00	1.76
3005	0.67	3	3514	2012.24	2013.24	1	2014.24	1.00	1.74
3202	0.67	3	3743	2142.75	2143.75	1	2144.75	1.00	1.75
3401	0.68	3	4014	2314.33	2315.33	1	2316.33	1.00	1.73
3603	0.70	3	4312	2510.55	2511.55	1	2512.55	1.00	1.72
3802	0.71	3	4597	2696.24	2697.24	1	2698.24	1.00	1.70
4002	0.73	3	4905	2904.16	2905.16	1	2906.16	1.00	1.69
4211	0.68	3	4950	2845.10	2846.10	1	2847.10	1.00	1.74
4401	0.67	3	5158	2957.72	2958.72	1	2959.72	1.00	1.74
4602	0.68	3	5406	3105.94	3106.94	1	3107.94	1.00	1.74
4806	0.67	3	5610	3207.13	3208.13	1	3209.13	1.00	1.75
5002	0.67	3	5830	3329.27	3330.27	1	3331.27	1.00	1.75
5199	0.66	3	6029	3429.75	3430.75	1	3431.75	1.00	1.76
5399	0.67	3	6312	3612.85	3613.85	1	3614.85	1.00	1.75
5598	0.66	3	6484	3685.67	3686.67	1	3687.67	1.00	1.76
5812	0.67	3	6784	3878.32	3879.32	1	3880.32	1.00	1.75
6005	0.67	3	7040	4037.81	4038.81	1	4039.81	1.00	1.74
6202	0.65	3	7154	4053.78	4054.78	1	4055.78	1.00	1.76
6396	0.67	3	7496	4298.57	4299.57	1	4300.57	1.00	1.74
6614	0.67	3	7730	4423.68	4424.68	1	4425.68	1.00	1.75
6802	0.64	3	7735	4334.66	4335.66	1	4336.66	1.00	1.78
7005	0.67	3	8168	4666.28	4667.28	1	4668.28	1.00	1.75
7205	0.67	3	8428	4825.87	4826.87	1	4827.87	1.00	1.75
7410	0.67	3	8682	4977.99	4978.99	1	4979.99	1.00	1.74
7602	0.67	3	8889	5088.70	5089.70	1	5090.70	1.00	1.75
7803	0.67	3	9121	5220.17	5221.17	1	5222.17	1.00	1.75
8003	0.67	3	9343	5341.90	5342.90	1	5343.90	1.00	1.75
8218	0.66	3	9565	5456.47	5457.47	1	5458.47	1.00	1.75
8411	0.66	3	9747	5542.11	5543.11	1	5544.11	1.00	1.76
8601	0.66	3	9958	5658.03	5659.03	1	5660.03	1.00	1.76
8802	0.67	3	10259	5858.28	5859.28	1	5860.28	1.00	1.75
9006	0.66	3	10490	5987.50	5988.50	1	5989.50	1.00	1.75
9202	0.67	3	10763	6162.99	6163.99	1	6164.99	1.00	1.75
9396	0.66	3	10874	6176.93	6177.93	1	6178.93	1.00	1.76
9603	0.65	3	11078	6276.55	6277.55	1	6278.55	1.00	1.76
9802	0.66	3	11396	6495.12	6496.12	1	6497.12	1.00	1.75
9998	0.66	3	11548	6549.71	6550.71	1	6551.71	1.00	1.76

Table A.6: Comparison of the upper bound (Section 4.1) and the lower bound (Section 4.2) for parallel exponentiation for the weights of Table A.2.

Addition chains with weighted length: sparse modulus									
Param.			binary	lower bound			Algorithm 4.12		
n	c	P	av. L	$e = 2^{n-1}$	$e = 2^n - 1$	Δ	depth	Δ	spd.
209	0.48	4	204	99.68	100.68	1	102.68	2.00	1.98
398	0.29	5	314	115.55	117.55	2	118.55	1.00	2.65
606	0.18	7	411	108.04	111.04	3	112.04	1.00	3.66
803	0.18	7	547	146.23	149.23	3	150.23	1.00	3.64
1018	0.19	7	705	196.46	198.46	2	200.46	2.00	3.52
1199	0.11	10	734	135.28	138.28	3	140.28	2.00	5.23
1401	0.10	11	847	146.80	149.80	3	151.80	2.00	5.58
1601	0.10	12	960	159.83	162.83	3	164.83	2.00	5.82
1791	0.10	11	1079	183.56	186.56	3	188.56	2.00	5.72
1996	0.10	11	1206	208.54	211.54	3	213.54	2.00	5.65
2212	0.08	14	1277	171.65	175.65	4	176.65	1.00	7.23
2406	0.08	13	1405	202.14	206.14	4	207.14	1.00	6.78
2613	0.08	14	1517	211.10	215.10	4	216.10	1.00	7.02
2802	0.08	14	1619	218.99	222.99	4	223.99	1.00	7.23
3005	0.08	14	1741	238.78	242.78	4	243.78	1.00	7.14
3202	0.08	15	1843	242.40	246.40	4	247.40	1.00	7.45
3401	0.06	17	1919	218.87	222.87	4	223.87	1.00	8.57
3603	0.08	15	2077	276.23	280.23	4	281.23	1.00	7.39
3802	0.08	15	2190	289.92	293.92	4	294.92	1.00	7.43
4002	0.08	14	2316	315.08	319.08	4	320.08	1.00	7.23
4211	0.07	17	2383	277.72	281.72	4	282.72	1.00	8.43
4401	0.05	22	2419	219.29	223.29	4	225.29	2.00	10.74
4602	0.05	22	2524	223.79	227.79	4	229.79	2.00	10.99
4806	0.05	23	2623	220.67	224.67	4	226.67	2.00	11.57
5002	0.05	20	2771	270.40	274.40	4	276.40	2.00	10.03
5199	0.04	24	2827	228.29	233.29	5	234.29	1.00	12.07
5399	0.04	25	2931	231.86	236.86	5	237.86	1.00	12.32
5598	0.04	25	3039	240.61	245.61	5	246.61	1.00	12.32
5812	0.04	25	3153	247.24	252.24	5	253.24	1.00	12.45
6005	0.05	21	3311	308.89	312.89	4	314.89	2.00	10.51
6202	0.04	25	3364	263.83	268.83	5	269.83	1.00	12.47
6396	0.04	26	3464	266.16	271.16	5	272.16	1.00	12.73
6614	0.04	26	3581	274.51	279.51	5	280.51	1.00	12.77
6802	0.05	22	3734	333.37	337.37	4	339.37	2.00	11.00
7005	0.05	22	3852	349.84	353.84	4	355.84	2.00	10.82
7205	0.05	22	3953	351.11	355.11	4	357.11	2.00	11.07
7410	0.04	25	4013	308.89	313.89	5	314.89	1.00	12.75
7602	0.04	25	4123	322.75	327.75	5	328.75	1.00	12.54
7803	0.05	21	4302	400.58	404.58	4	406.58	2.00	10.58
8003	0.05	21	4422	420.58	424.58	4	426.58	2.00	10.37
8218	0.04	26	4449	340.79	345.79	5	346.79	1.00	12.83
8411	0.04	25	4565	360.18	365.18	5	366.18	1.00	12.47
8601	0.03	31	4596	296.45	301.45	5	302.45	1.00	15.20
8802	0.03	32	4688	287.12	292.12	5	293.12	1.00	15.99
9006	0.03	33	4789	286.93	291.93	5	292.93	1.00	16.35
9202	0.03	33	4892	291.26	296.26	5	297.26	1.00	16.46
9396	0.03	35	4981	283.71	288.71	5	290.71	2.00	17.13
9603	0.04	29	5145	344.31	349.31	5	350.31	1.00	14.69
9802	0.03	30	5242	341.35	346.35	5	347.35	1.00	15.09
9998	0.03	35	5296	297.36	302.36	5	304.36	2.00	17.40

Table A.7: Comparison of the upper bound (Section 4.1) and the lower bound (Section 4.2) for parallel exponentiation for the weights of Table A.3.

Addition chains with weighted length: sedimentary modulus									
Param.			binary	lower bound			Algorithm 4.12		
n	c	P	av. L	$e = 2^{n-1}$	$e = 2^n - 1$	Δ	depth	Δ	spd.
209	0.57	3	223	118.94	119.94	1	120.94	1.00	1.84
398	0.32	5	326	127.87	129.87	2	130.87	1.00	2.49
606	0.24	6	449	146.47	148.47	2	150.47	2.00	2.98
803	0.24	6	592	190.99	192.99	2	194.99	2.00	3.04
1018	0.21	6	719	210.03	212.03	2	214.03	2.00	3.36
1199	0.15	8	781	182.43	185.43	3	186.43	1.00	4.19
1401	0.14	9	894	194.36	197.36	3	198.36	1.00	4.51
1601	0.16	8	1049	249.04	252.04	3	253.04	1.00	4.15
1791	0.13	9	1135	239.97	242.97	3	243.97	1.00	4.65
1996	0.13	9	1258	260.68	263.68	3	264.68	1.00	4.75
2212	0.11	10	1352	246.47	249.47	3	251.47	2.00	5.38
2406	0.10	11	1453	250.01	253.01	3	255.01	2.00	5.70
2613	0.09	12	1544	238.48	242.48	4	243.48	1.00	6.34
2802	0.08	13	1635	234.47	238.47	4	239.47	1.00	6.83
3005	0.09	13	1770	268.34	272.34	4	273.34	1.00	6.48
3202	0.09	12	1899	298.10	301.10	3	303.10	2.00	6.26
3401	0.10	11	2047	346.78	349.78	3	351.78	2.00	5.82
3603	0.09	13	2111	309.97	313.97	4	314.97	1.00	6.70
3802	0.08	14	2205	304.18	308.18	4	309.18	1.00	7.13
4002	0.10	12	2389	388.28	391.28	3	393.28	2.00	6.07
4211	0.07	15	2420	314.59	318.59	4	319.59	1.00	7.57
4401	0.07	16	2494	293.86	297.86	4	298.86	1.00	8.34
4602	0.06	18	2584	283.47	287.47	4	289.47	2.00	8.93
4806	0.07	16	2730	327.48	331.48	4	332.48	1.00	8.21
5002	0.07	16	2840	339.16	343.16	4	344.16	1.00	8.25
5199	0.06	18	2905	306.36	310.36	4	312.36	2.00	9.30
5399	0.06	19	3009	309.95	313.95	4	315.95	2.00	9.52
5598	0.05	20	3101	303.00	307.00	4	309.00	2.00	10.04
5812	0.05	20	3218	312.17	316.17	4	318.17	2.00	10.11
6005	0.06	19	3350	348.32	352.32	4	354.32	2.00	9.46
6202	0.05	20	3432	331.13	335.13	4	337.13	2.00	10.18
6396	0.05	21	3532	334.66	338.66	4	340.66	2.00	10.37
6614	0.05	21	3652	345.54	349.54	4	351.54	2.00	10.39
6802	0.05	21	3755	354.92	358.92	4	360.92	2.00	10.41
7005	0.06	19	3892	390.04	394.04	4	396.04	2.00	9.83
7205	0.07	16	4100	498.49	502.49	4	503.49	1.00	8.14
7410	0.05	20	4097	392.70	396.70	4	398.70	2.00	10.28
7602	0.05	20	4209	408.41	412.41	4	414.41	2.00	10.16
7803	0.06	19	4350	449.30	453.30	4	455.30	2.00	9.55
8003	0.07	15	4576	575.19	579.19	4	580.19	1.00	7.89
8218	0.05	21	4534	425.23	429.23	4	431.23	2.00	10.51
8411	0.05	22	4620	415.22	419.22	4	421.22	2.00	10.97
8601	0.05	23	4697	396.73	400.73	4	402.73	2.00	11.66
8802	0.05	22	4830	429.47	433.47	4	435.47	2.00	11.09
9006	0.04	27	4862	359.19	364.19	5	365.19	1.00	13.31
9202	0.04	27	4966	365.13	370.13	5	371.13	1.00	13.38
9396	0.04	28	5057	359.58	364.58	5	365.58	1.00	13.83
9603	0.05	22	5278	476.93	480.93	4	482.93	2.00	10.93
9802	0.04	24	5336	435.39	440.39	5	441.39	1.00	12.09
9998	0.04	28	5372	373.89	378.89	5	379.89	1.00	14.14

Table A.8: Comparison of the upper bound (Section 4.1) and the lower bound (Section 4.2) for parallel exponentiation for the weights of Table A.4.

A.2. Basic arithmetic.

A.2.1. Polynomial basis representation. Basic arithmetic for $\mathbb{F}_{2^n}$ in a polynomial basis representation is discussed in Section 5. The times for the three different moduli are listed in separate tables. All times are the average of 10000 trials. Table A.9 contains the figures for arbitrary modulus f, i.e. the test series *Arbitrary*. Table A.10 lists the times for the test series *Sparse*. Here the moduli are trinomials and pentanomials. The third test series *Sedimentary* includes irreducible sedimentary polynomials. The experiments are documented in Table A.11. All three tables have the same structure, but column 2 is missed out in Table A.9.

The key to Tables A.9–A.11		
column	label	description
1–2	Param.	characterization of the modulus f
1	n	the degree of f
2	$—/e_3, e_2, e_1/k$	*Table A.9:* no such column, *Table A.10:* indices of the non-negative coefficients in the modulus: only e_1 given: $f = x^n + x^{e_1} + 1 \in \mathbb{F}_2[x]$ is a trinomial. Else $f = x^n + x^{e_3} + x^{e_2} + x^{e_1} + 1 \in \mathbb{F}_2[x]$. *Table A.11:* k is the degree of the sediment h and $f = x^n + h \in \mathbb{F}_2[x]$.
3–4	multiplication	figures for the multiplication of two randomly chosen elements of $\mathbb{F}_{2^n}$
3	c_A	the average time over 10000 trials in CPU-milliseconds
4	#poly	the number of multiplications of randomly chosen polynomials in $\mathbb{F}_2[x]$ of degree less than n that can be computed in the same time
5–6	squaring	figures for the squaring of a randomly chosen element of $\mathbb{F}_{2^n}$
5	c_Q	analogous column 3
6	#poly	as column 4
7	c_Q/c_A	the quotient $c = \frac{c_Q}{c_A}$ of col. 5 and col. 3
8–9	inverse	figures for the inversion of a randomly chosen element of $\mathbb{F}_{2^n}$
8	—	the average time over 1000 trials in CPU-milliseconds
9	#poly	as column 4

Arithmetic in $\mathbb{F}_2[x]/(f)$							
Para.	multiplication		squaring			inversion	
n	c_A	#poly	c_Q	#poly	c_Q/c_A		#poly
209	0.05	2.60	0.04	1.93	0.74	0.15	7.32
398	0.14	2.32	0.09	1.49	0.64	0.38	6.13
606	0.27	2.21	0.17	1.34	0.61	0.74	5.97
803	0.42	2.45	0.27	1.57	0.64	1.18	6.82
1018	0.56	2.81	0.39	1.95	0.69	1.78	8.87
1199	0.83	2.41	0.52	1.49	0.62	2.40	6.93
1401	1.05	2.48	0.66	1.56	0.63	3.15	7.44
1601	1.31	2.60	0.84	1.68	0.64	3.99	7.93
1791	1.52	2.81	1.02	1.88	0.67	4.93	9.10
1996	1.78	3.05	1.25	2.13	0.70	6.02	10.28
2212	2.33	2.64	1.50	1.70	0.64	7.29	8.28
2406	2.70	2.64	1.73	1.69	0.64	8.50	8.32
2613	3.09	2.68	2.00	1.73	0.65	9.87	8.55
2802	3.45	2.75	2.26	1.80	0.65	11.28	8.98
3005	3.82	2.86	2.56	1.92	0.67	12.86	9.63
3202	4.29	2.85	2.87	1.91	0.67	14.51	9.63
3401	4.74	2.99	3.22	2.04	0.68	16.28	10.29
3603	5.14	3.13	3.58	2.18	0.70	18.19	11.09
3802	5.54	3.24	3.93	2.30	0.71	20.17	11.79
4002	6.00	3.44	4.36	2.50	0.73	22.36	12.83
4211	6.54	2.99	4.42	2.02	0.68	24.69	11.27
4401	7.52	2.95	5.06	1.98	0.67	26.77	10.48
4602	8.14	2.98	5.50	2.01	0.68	29.50	10.79
4806	8.86	2.90	5.91	1.94	0.67	31.94	10.46
5002	9.33	2.92	6.21	1.95	0.67	34.37	10.77
5199	10.06	2.93	6.64	1.93	0.66	37.60	10.97
5399	10.63	2.91	7.12	1.95	0.67	39.96	10.95
5598	10.97	2.92	7.22	1.92	0.66	42.69	11.36
5812	11.48	2.93	7.66	1.96	0.67	45.65	11.65
6005	11.89	2.97	8.00	2.00	0.67	48.88	12.20
6202	12.40	2.96	8.11	1.93	0.65	51.87	12.37
6396	13.01	2.96	8.75	1.99	0.67	55.28	12.59
6614	13.40	2.93	8.97	1.96	0.67	59.15	12.94
6802	13.76	2.92	8.77	1.86	0.64	62.31	13.20
7005	14.05	2.92	9.36	1.95	0.67	66.00	13.71
7205	14.51	2.95	9.72	1.98	0.67	70.33	14.31
7410	14.86	2.92	9.98	1.96	0.67	74.40	14.62
7602	15.04	2.96	10.07	1.98	0.67	77.83	15.29
7803	15.31	2.96	10.25	1.98	0.67	81.49	15.74
8003	15.42	2.95	10.30	1.97	0.67	85.40	16.33
8218	16.82	2.97	11.17	1.97	0.66	90.27	15.92
8411	19.34	2.96	12.74	1.95	0.66	94.63	14.49
8601	20.96	2.94	13.79	1.94	0.66	99.02	13.90
8802	22.65	2.96	15.07	1.97	0.67	102.81	13.42
9006	23.71	2.95	15.76	1.96	0.66	108.36	13.51
9202	24.52	2.99	16.43	2.00	0.67	113.09	13.79
9396	25.97	2.96	17.08	1.95	0.66	117.48	13.38
9603	26.80	2.93	17.52	1.92	0.65	123.63	13.52
9802	27.59	2.93	18.28	1.94	0.66	127.78	13.57
9998	28.21	2.96	18.48	1.94	0.66	133.09	13.95

Table A.9: Times for basic operations (multiplication, squaring, inversion) in $\mathbb{F}_{2^n}$ for the test series *Arbitrary*.

Arithmetik in $\mathbb{F}_2[x]/(x^n + x^{e_3} + x^{e_2} + x^{e_1} + 1)$								
Parameter		multiplication		squaring			inversion	
n	e_3, e_2, e_1	c_A	#poly	c_Q	#poly	c_Q/c_A		#poly
209	6	0.03	1.32	0.01	0.63	0.48	0.15	7.26
398	7, 6, 2	0.07	1.18	0.02	0.34	0.29	0.38	6.14
606	165	0.13	1.06	0.02	0.19	0.18	0.75	6.05
803	14, 9, 2	0.19	1.08	0.03	0.20	0.18	1.19	6.72
1018	12, 10, 5	0.22	1.09	0.04	0.21	0.19	1.80	9.11
1199	114	0.36	1.03	0.04	0.12	0.11	2.39	6.91
1401	92	0.44	1.03	0.05	0.11	0.10	3.16	7.47
1601	548	0.52	1.02	0.05	0.10	0.10	3.99	7.91
1791	190	0.56	1.03	0.06	0.11	0.10	4.88	8.99
1996	307	0.60	1.03	0.06	0.11	0.10	6.01	10.28
2212	423	0.90	1.02	0.07	0.08	0.08	7.28	8.27
2406	14, 12, 5	1.05	1.03	0.09	0.09	0.08	8.52	8.34
2613	14, 12, 9	1.19	1.03	0.10	0.08	0.08	9.89	8.56
2802	18, 14, 3	1.29	1.03	0.10	0.08	0.08	11.42	9.09
3005	14, 12, 5	1.37	1.03	0.11	0.08	0.08	13.06	9.75
3202	24, 12, 3	1.54	1.03	0.12	0.08	0.08	14.62	9.79
3401	531	1.60	1.01	0.10	0.07	0.06	16.43	10.41
3603	23, 11, 6	1.69	1.03	0.13	0.08	0.08	18.29	11.15
3802	16, 15, 8	1.77	1.05	0.14	0.08	0.08	20.24	11.93
4002	24, 11, 6	1.80	1.04	0.14	0.08	0.08	22.35	12.84
4211	14, 12, 6	2.25	1.03	0.15	0.07	0.07	24.66	11.24
4401	394	2.60	1.01	0.13	0.05	0.05	26.83	10.39
4602	675	2.77	1.01	0.13	0.05	0.05	29.44	10.69
4806	2349	3.08	1.01	0.14	0.05	0.05	31.69	10.42
5002	39, 23, 5	3.25	1.02	0.18	0.06	0.05	34.24	10.72
5199	1546	3.46	1.01	0.15	0.04	0.04	36.95	10.76
5399	485	3.67	1.01	0.16	0.04	0.04	39.80	10.97
5598	101	3.78	1.01	0.16	0.04	0.04	42.71	11.39
5812	295	3.96	1.01	0.17	0.04	0.04	45.77	11.70
6005	28, 12, 2	4.06	1.02	0.21	0.05	0.05	48.83	12.23
6202	867	4.23	1.01	0.18	0.04	0.04	52.84	12.60
6396	91	4.43	1.01	0.18	0.04	0.04	55.39	12.61
6614	2105	4.61	1.01	0.19	0.04	0.04	59.15	12.94
6802	29, 25, 3	4.82	1.02	0.24	0.05	0.05	62.43	13.25
7005	14, 11, 4	4.89	1.01	0.24	0.05	0.05	66.28	13.71
7205	21, 5, 2	5.00	1.01	0.24	0.05	0.05	69.47	14.06
7410	2179	5.10	1.02	0.21	0.04	0.04	73.68	14.70
7602	555	5.13	1.01	0.22	0.04	0.04	77.53	15.26
7803	19, 14, 2	5.25	1.02	0.27	0.05	0.05	81.64	15.81
8003	26, 21, 2	5.31	1.02	0.28	0.05	0.05	86.09	16.52
8218	1443	5.68	1.01	0.24	0.04	0.04	90.57	16.05
8411	30, 27, 5	6.62	1.01	0.28	0.04	0.04	94.91	14.53
8601	734	7.15	1.01	0.25	0.03	0.03	99.72	14.01
8802	2139	7.71	1.01	0.25	0.03	0.03	103.79	13.55
9006	1477	8.07	1.01	0.26	0.03	0.03	109.31	13.62
9202	211	8.29	1.01	0.26	0.03	0.03	113.28	13.81
9396	369	8.87	1.01	0.27	0.03	0.03	118.31	13.50
9603	19, 10, 4	9.21	1.01	0.33	0.04	0.04	123.10	13.45
9802	12, 10, 3	9.47	1.01	0.33	0.04	0.03	129.32	13.77
9998	4013	9.60	1.01	0.29	0.03	0.03	132.49	13.88

Table A.10: Times for basic operations (multiplication, squaring, inversion) in $\mathbb{F}_{2^n}$ for the test series *Sparse*.

Arithmetik in $\mathbb{F}_2[x]/(x^n + h)$ with $\deg h = k$								
Parameter		multiplication		squaring			inversion	
n	k	c_A	#poly	c_Q	#poly	c_Q/c_A		#poly
209	5	0.03	1.57	0.02	0.90	0.57	0.15	7.29
398	7	0.08	1.24	0.02	0.40	0.32	0.39	6.35
606	9	0.14	1.15	0.03	0.28	0.24	0.73	5.91
803	8	0.20	1.18	0.05	0.28	0.24	1.19	6.89
1018	10	0.22	1.12	0.05	0.23	0.21	1.77	8.84
1199	11	0.37	1.08	0.06	0.16	0.15	2.40	6.93
1401	11	0.46	1.08	0.06	0.15	0.14	3.14	7.42
1601	11	0.56	1.11	0.09	0.17	0.16	3.97	7.88
1791	12	0.58	1.07	0.08	0.14	0.13	4.94	9.08
1996	9	0.63	1.07	0.08	0.14	0.13	5.98	10.23
2212	11	0.95	1.08	0.11	0.12	0.11	7.27	8.27
2406	8	1.10	1.07	0.11	0.11	0.10	8.57	8.38
2613	11	1.21	1.05	0.11	0.10	0.09	10.01	8.67
2802	9	1.31	1.05	0.11	0.09	0.08	11.29	9.00
3005	9	1.39	1.04	0.12	0.09	0.09	12.88	9.63
3202	9	1.59	1.07	0.15	0.10	0.09	14.65	9.84
3401	11	1.67	1.06	0.17	0.11	0.10	16.28	10.33
3603	10	1.71	1.04	0.15	0.09	0.09	18.17	11.07
3802	13	1.81	1.07	0.14	0.09	0.08	20.20	11.92
4002	8	1.86	1.07	0.18	0.10	0.10	22.32	12.85
4211	12	2.28	1.03	0.17	0.08	0.07	24.64	11.18
4401	12	2.65	1.03	0.18	0.07	0.07	26.84	10.45
4602	14	2.82	1.03	0.17	0.06	0.06	29.22	10.63
4806	12	3.18	1.04	0.22	0.07	0.07	31.73	10.39
5002	11	3.33	1.04	0.23	0.07	0.07	34.66	10.86
5199	12	3.52	1.03	0.21	0.06	0.06	37.25	10.86
5399	9	3.75	1.03	0.22	0.06	0.06	39.73	10.94
5598	9	3.85	1.02	0.21	0.06	0.05	43.03	11.45
5812	11	4.02	1.03	0.22	0.06	0.05	45.95	11.75
6005	12	4.09	1.03	0.24	0.06	0.06	49.23	12.34
6202	12	4.31	1.03	0.23	0.05	0.05	52.04	12.39
6396	12	4.50	1.02	0.24	0.05	0.05	55.73	12.68
6614	12	4.69	1.03	0.25	0.05	0.05	59.05	12.96
6802	11	4.84	1.03	0.25	0.05	0.05	62.88	13.34
7005	13	4.93	1.03	0.27	0.06	0.06	66.15	13.76
7205	14	5.12	1.04	0.35	0.07	0.07	69.52	14.13
7410	10	5.15	1.02	0.27	0.05	0.05	73.63	14.63
7602	10	5.22	1.03	0.28	0.06	0.05	77.49	15.25
7803	12	5.30	1.03	0.31	0.06	0.06	81.69	15.82
8003	8	5.44	1.04	0.39	0.07	0.07	85.83	16.40
8218	12	5.81	1.03	0.30	0.05	0.05	90.52	16.04
8411	12	6.66	1.02	0.33	0.05	0.05	94.65	14.44
8601	7	7.27	1.02	0.34	0.05	0.05	99.34	13.98
8802	14	7.90	1.03	0.39	0.05	0.05	103.38	13.48
9006	9	8.19	1.02	0.33	0.04	0.04	108.14	13.48
9202	12	8.44	1.03	0.33	0.04	0.04	113.33	13.79
9396	13	8.92	1.01	0.34	0.04	0.04	118.32	13.40
9603	12	9.40	1.03	0.47	0.05	0.05	123.99	13.62
9802	12	9.61	1.02	0.43	0.05	0.04	129.63	13.81
9998	13	9.71	1.02	0.36	0.04	0.04	132.88	13.93

Table A.11: Times for basic operations (multiplication, squaring, inversion) in $\mathbb{F}_{2^n}$ for the test series *Sedimentary*.

A.2.2. Normal basis representation. Multiplication, squaring and inversion in a normal basis representation of $\mathbb{F}_{2^n}$ is presented in detail in Section 6. Test series *Normal* uses a matrix-based multiplication algorithm and works for arbitrary normal bases. The experiments are analogous to those for the polynomial basis representations but there are only 1000 trials for each value of n. Table A.12 has a similar structure as Tables A.9–A.11.

The key to Table A.12		
column	label	description
1–2	Param.	parameter of the optimal normal basis
1	n	the degree of the field extension over $\mathbb{F}_2$
2	k	the type of the optimal normal basis
3–4	multiplication	figures for the multiplication of two randomly chosen elements in $\mathbb{F}_{2^n}$
3	c_A	the average time over 1000 trials in CPU-milliseconds
4	#poly	the number of multiplications of randomly chosen polynomials in $\mathbb{F}_2[x]$ of degree less than n that can be computed in the same time
5–6	squaring	analogous columns 3–4 for squaring
7	c_Q/c_A	the quotient $c = \frac{c_Q}{c_A}$ of col. 5 and col. 3

Arithmetic using a normal basis $\mathcal{N}$ of type k						
Parameter		multiplication		squaring		
n	k	c_A	#poly	c_Q	#poly	c_Q/c_A
209	2	1.94	97.32	0.00	0.12	0.00
398	2	6.97	114.80	0.00	0.05	0.00
606	2	16.03	131.57	0.00	0.03	0.00
803	2	28.03	163.43	0.00	0.02	0.00
1018	1	33.10	166.88	0.01	0.03	0.00
1199	2	62.64	182.44	0.01	0.02	0.00
1401	2	85.31	202.67	0.01	0.02	0.00
1601	2	112.24	224.60	0.01	0.01	0.00
1791	2	140.39	260.43	0.01	0.01	0.00
1996	1	126.51	217.65	0.01	0.01	0.00
2212	1	155.93	178.08	0.01	0.01	0.00
2406	2	269.16	264.72	0.01	0.01	0.00
2613	2	324.44	282.29	0.01	0.01	0.00
2802	1	252.03	201.66	0.01	0.01	0.00
3005	2	432.89	325.85	0.01	0.01	0.00
3202	1	330.39	220.24	0.01	0.01	0.00
3401	2	555.79	352.91	0.01	0.01	0.00
3603	2	622.50	380.88	0.01	0.01	0.00
3802	1	472.96	277.67	0.02	0.01	0.00
4002	1	525.73	303.13	0.01	0.01	0.00
4211	2	852.51	390.51	0.02	0.01	0.00
4401	2	930.89	365.73	0.02	0.01	0.00
4602	1	718.69	263.71	0.02	0.01	0.00
4806	2	1109.56	365.94	0.02	0.01	0.00
5002	1	854.73	268.62	0.02	0.01	0.00
5199	2	1297.36	379.43	0.02	0.01	0.00
5399	2	1399.88	384.67	0.02	0.01	0.00
5598	2	1504.22	401.28	0.02	0.01	0.00
5812	1	1161.42	297.36	0.02	0.01	0.00
6005	2	1730.87	433.38	0.02	0.01	0.00
6202	1	1324.32	316.75	0.02	0.01	0.00
6396	1	1404.63	320.66	0.02	0.01	0.00
6614	2	2100.66	460.98	0.02	0.01	0.00
6802	1	1589.78	337.70	0.02	0.01	0.00
7005	2	2354.07	490.48	0.03	0.01	0.00
7205	2	2490.07	508.20	0.03	0.01	0.00
7410	1	1887.56	372.03	0.03	0.01	0.00
7602	1	1982.59	390.77	0.03	0.00	0.00
7803	2	2920.99	565.91	0.03	0.01	0.00
8003	2	3074.39	589.68	0.03	0.01	0.00
8218	1	2321.12	410.50	0.03	0.01	0.00
8411	2	3395.38	521.59	0.02	0.00	0.00
8601	2	3557.05	500.51	0.03	0.00	0.00
8802	1	2659.76	348.05	0.03	0.00	0.00
9006	2	3892.01	486.12	0.03	0.00	0.00
9202	1	2906.29	355.14	0.03	0.00	0.00
9396	1	3031.63	346.03	0.03	0.00	0.00
9603	2	4430.00	485.61	0.03	0.00	0.00
9802	1	3298.51	350.93	0.03	0.00	0.00
9998	2	4795.50	503.80	0.04	0.00	0.00

Table A.12: Times for basic operations (multiplication, squaring) in $\mathbb{F}_{2^n}$ for test series *Normal*.

The times for inversion are the average of 100 trials. They are listed separately in Table A.13.

The key to Table A.13		
column	label	description
1–2	Param.	as columns 1–2 of Table A.12
3–10	in sequential	figures for sequential inversion
3–4	binary	input addition chain for $n-1$ is the binary addition chain
3	t	the average time over 100 trials in CPU-milliseconds
4	L	the length of the input addition chain for $n-1$
5–7	Brauer	input addition chain for $n-1$ is a Brauer addition chain
5–6	t, L	as columns 3–4
7	Δ	the quotient of col. 5 and col. 3
8–10	power tree	input addition chain for $n-1$ is a a power tree addition chain
8–9	t, L	as columns 3–4
10	Δ	the quotient of col. 8 and col. 3
11–12	in parallel	figures for parallel inversion on two processors, see Algorithm 6.27
11	t	the average time over 100 trials in CPU-milliseconds
12	Δ	the quotient of col. 11 and col. 3

Inversion a la Fermat using a normal basis $\mathcal{N}$ of type k												
Parameter		in sequential								in parallel		
n	k	binary		Brauer			power tree					
		t	L	t	L	Δ	t	L	Δ	t	Δ	
209	2	17.09	10	17.59	10	0.97	17.55	10	0.97	15.95	1.07	
398	2	86.94	13	83.82	13	1.04	76.88	12	1.13	63.22	1.38	
606	2	230.01	15	225.20	15	1.02	192.90	13	1.19	161.29	1.43	
803	2	338.75	13	337.53	13	1.00	338.10	13	1.00	282.53	1.20	
1018	1	525.57	17	463.62	15	1.13	433.11	14	1.21	334.27	1.57	
1199	2	936.99	16	942.37	16	0.99	812.83	14	1.15	687.66	1.36	
1401	2	1271.67	16	1285.84	16	0.99	1115.64	14	1.14	942.03	1.35	
1601	2	1358.70	13	1345.21	13	1.01	1343.80	13	1.01	1231.01	1.10	
1791	2	2509.80	19	2252.12	17	1.11	1980.57	15	1.27	1561.11	1.61	
1996	1	2131.47	18	1900.24	16	1.12	1772.61	15	1.20	1397.30	1.53	
2212	1	2344.81	16	2336.14	16	1.00	2174.42	15	1.08	1866.25	1.26	
2406	2	4334.40	17	4313.31	17	1.00	3771.90	15	1.15	3238.25	1.34	
2613	2	4877.26	16	4851.60	16	1.01	4560.15	15	1.07	3891.06	1.25	
2802	1	4279.48	18	4281.14	18	1.00	3788.14	16	1.13	3012.38	1.42	
3005	2	7741.47	19	7369.85	18	1.05	6494.35	16	1.19	5203.05	1.49	
3202	1	4527.47	15	4607.72	15	0.98	4613.07	15	0.98	3956.02	1.14	
3401	2	8402.38	16	8346.91	16	1.01	7790.34	15	1.08	6674.10	1.26	
3603	2	9434.85	16	9353.21	16	1.01	9366.45	16	1.01	7493.37	1.26	
3802	1	8530.44	19	7997.75	18	1.07	7578.98	17	1.13	5685.95	1.50	
4002	1	8947.48	18	8405.96	17	1.06	7900.36	16	1.13	6325.21	1.41	
4211	2	13637.00	17	13634.30	17	1.00	12783.90	16	1.07	11079.90	1.23	
4401	2	14013.00	16	13955.30	16	1.00	13946.50	16	1.00	12107.40	1.16	
4602	1	13570.50	20	12709.80	19	1.07	11314.00	17	1.20	9274.14	1.46	
4806	2	18810.20	18	18872.50	18	1.00	16661.60	16	1.13	14404.80	1.31	
5002	1	14673.20	18	14508.20	18	1.01	12799.20	16	1.15	11098.20	1.32	
5199	2	21999.80	18	22107.70	18	1.00	20799.70	17	1.06	16867.70	1.30	
5399	2	23374.80	18	23805.20	18	0.98	21011.00	16	1.11	18190.20	1.29	
5598	2	30274.60	21	28577.00	20	1.06	24094.20	17	1.26	19561.50	1.55	
5812	1	22154.20	20	20892.20	19	1.06	18573.20	17	1.19	15089.20	1.47	
6005	2	32644.10	20	31195.90	19	1.05	27729.80	17	1.18	22495.40	1.45	
6202	1	22579.20	18	22477.60	18	1.00	21162.80	17	1.07	17171.20	1.31	
6396	1	28244.00	21	25270.00	19	1.12	22480.70	17	1.26	18288.80	1.54	
6614	2	40048.40	20	39948.60	20	1.00	33640.60	17	1.19	27324.10	1.47	
6802	1	26944.10	18	27044.20	18	1.00	25471.40	17	1.06	20665.80	1.30	
7005	2	44998.50	20	42465.10	19	1.06	37708.60	17	1.19	30644.50	1.47	
7205	2	39998.30	17	39851.50	17	1.00	39997.20	17	1.00	32457.70	1.23	
7410	1	35855.60	20	32061.90	18	1.12	32134.60	18	1.12	24524.30	1.46	
7602	1	37704.50	20	35749.60	19	1.05	33775.50	18	1.12	25829.10	1.46	
7803	2	58726.20	21	55472.40	20	1.06	46736.00	17	1.26	38076.30	1.54	
8003	2	55496.70	19	52336.60	18	1.06	49197.90	17	1.13	40070.70	1.38	
8218	1	37214.50	17	37115.90	17	1.00	37092.80	17	1.00	32458.80	1.15	
8411	2	61220.00	19	61120.40	19	1.00	57736.50	18	1.06	47598.40	1.29	
8601	2	61083.60	18	56790.10	17	1.08	56822.90	17	1.07	49715.90	1.23	
8802	1	45079.10	18	45282.20	18	1.00	45247.90	18	1.00	37262.00	1.21	
9006	2	74419.00	20	74050.60	20	1.00	66143.70	18	1.13	54524.00	1.36	
9202	1	58281.10	21	55245.90	20	1.05	49417.30	18	1.18	40753.80	1.43	
9396	1	57053.70	20	57607.10	20	0.99	51534.50	18	1.11	42455.40	1.34	
9603	2	75584.80	18	75292.60	18	1.00	70754.60	17	1.07	62076.10	1.22	
9802	1	59151.00	19	59355.90	19	1.00	52794.30	17	1.12	46259.90	1.28	
9998	2	90891.80	20	91102.20	20	1.00	76661.80	17	1.19	67175.20	1.35	

Table A.13: Times for inversion in $\mathbb{F}_{2^n}$ for test series *Normal*.

A.2.3. Normal Gauß periods. The algorithms for normal Gauß periods are developed in Section 8 for prime power Gauß periods and in Section 9 for decomposable Gauß periods. The test series *PrimeGP* contains the same fields as the test series *Normal* but uses fast polynomial multiplication instead of matrix-based multiplication. The experiments documented in Table A.14 are analogous to those given in Table A.12. But for *PrimeGP* the times are the average of 10000 trials for multiplication and squaring.

The key to Table A.14		
column	label	description
1–2	Param.	parameter of the normal prime Gauß period
1	n	the degree of the field extension over $\mathbb{F}_2$
2	k	the order of the subgroup $\mathcal{K} \subseteq \mathbb{Z}_{nk+1}^{\times}$
3–4	multiplication	figures for the multiplication of two randomly chosen elements in $\mathbb{F}_{2^n}$
3	c_A	the average time over 10000 trials in CPU-milliseconds
4	#poly	the number of multiplications of randomly chosen polynomials in $\mathbb{F}_2[x]$ of degree less than n that can be computed in the same time
5–6	squaring	analogous columns 3–4 for squaring
7	c_Q/c_A	the quotient $c = \frac{c_Q}{c_A}$ of col. 5 and col. 3
8	vs. $T_{\mathcal{N}}$	the quotient of column 3 of this table and column 3 of Table A.12; it is the speed-up that is achieved by exchanging matrix-based multiplication by polynomial based multiplication for normal prime Gauß periods

Arithmetic using a normal Gauß period of type (n, k)							
Parameter		multiplication		squaring			vs. $T_{\mathcal{N}}$
n	k	c_A	#poly	c_Q	#poly	c_Q/c_A	
209	2	0.12	5.83	0.00	0.12	0.02	16.27
398	2	0.26	4.20	0.00	0.05	0.01	27.08
606	2	0.48	3.87	0.00	0.03	0.01	33.59
803	2	0.67	3.90	0.00	0.03	0.01	41.72
1018	1	0.36	1.82	0.01	0.03	0.01	91.03
1199	2	1.25	3.65	0.01	0.02	0.00	49.94
1401	2	1.54	3.64	0.01	0.02	0.00	55.55
1601	2	1.82	3.63	0.01	0.01	0.00	61.80
1791	2	1.98	3.66	0.01	0.01	0.00	71.01
1996	1	0.90	1.54	0.01	0.01	0.01	140.87
2212	1	1.23	1.40	0.01	0.01	0.01	127.12
2406	2	3.53	3.47	0.01	0.01	0.00	76.29
2613	2	3.99	3.47	0.01	0.01	0.00	81.40
2802	1	1.69	1.35	0.01	0.01	0.01	148.74
3005	2	4.60	3.46	0.01	0.01	0.00	94.09
3202	1	1.99	1.33	0.01	0.01	0.01	166.03
3401	2	5.41	3.43	0.01	0.01	0.00	102.82
3603	2	5.67	3.47	0.01	0.01	0.00	109.82
3802	1	2.29	1.34	0.01	0.01	0.01	206.41
4002	1	2.37	1.36	0.02	0.01	0.01	222.11
4211	2	7.46	3.42	0.02	0.01	0.00	114.24
4401	2	8.55	3.36	0.02	0.01	0.00	108.89
4602	1	3.46	1.27	0.02	0.01	0.00	207.59
4806	2	10.11	3.32	0.02	0.01	0.00	109.77
5002	1	3.98	1.25	0.02	0.01	0.00	214.52
5199	2	11.28	3.30	0.02	0.01	0.00	114.97
5399	2	11.99	3.29	0.02	0.01	0.00	116.77
5598	2	12.38	3.30	0.02	0.01	0.00	121.55
5812	1	4.83	1.24	0.02	0.01	0.00	240.69
6005	2	13.19	3.30	0.02	0.01	0.00	131.25
6202	1	5.18	1.24	0.02	0.01	0.00	255.70
6396	1	5.40	1.23	0.02	0.01	0.00	260.10
6614	2	15.03	3.30	0.02	0.01	0.00	139.73
6802	1	5.79	1.23	0.02	0.01	0.00	274.62
7005	2	15.86	3.30	0.03	0.01	0.00	148.43
7205	2	16.20	3.31	0.03	0.01	0.00	153.72
7410	1	6.19	1.22	0.03	0.01	0.00	304.76
7602	1	6.32	1.24	0.03	0.01	0.00	313.84
7803	2	17.07	3.30	0.03	0.01	0.00	171.16
8003	2	17.32	3.32	0.03	0.01	0.00	177.53
8218	1	6.94	1.23	0.03	0.01	0.00	334.45
8411	2	21.29	3.27	0.03	0.00	0.00	159.45
8601	2	23.14	3.26	0.03	0.00	0.00	153.71
8802	1	9.07	1.19	0.03	0.00	0.00	293.26
9006	2	25.93	3.24	0.03	0.00	0.00	150.07
9202	1	9.69	1.18	0.03	0.00	0.00	299.92
9396	1	10.29	1.17	0.03	0.00	0.00	294.59
9603	2	29.26	3.21	0.03	0.00	0.00	151.40
9802	1	10.88	1.16	0.03	0.00	0.00	303.07
9998	2	30.78	3.23	0.04	0.00	0.00	155.80

Table A.14: Times for basic operations (multiplication, squaring) in $\mathbb{F}_{2^n}$ for the test series *PrimeGP*.

Table A.15 documents the experiments on multiplication and squaring for the test series *PrimepowerGP*. Its goal is to give a comparison between prime Gauß periods and prime power Gauß periods as described in Section 8.1. All figures are the average of 10000 trials.

The key to Table A.15		
column	label	description
1	n	the degree of the extension field $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$ for which normal prime and prime power Gauß periods are compared
2–6	prime power Gauß period	parameters and times for prime power Gauß periods of type $(n, \mathcal{K})$ over $\mathbb{F}_2$
2	k	the order of the subgroup $\mathcal{K}$
3	r	$\mathcal{K} \subseteq \mathbb{Z}_r^\times$ and $\phi(r) = nk$
4	c_A	the average time over 10000 multiplications in CPU-milliseconds
5	#poly	the number of multiplications of randomly chosen polynomials in $\mathbb{F}_2[x]$ of degree less than n that can be computed in the same time
6	c_Q	the average time over 10000 squarings in CPU-milliseconds
7–11	prime Gauß period	as columns 2–6 but for prime Gauß period of type (n, k) over $\mathbb{F}_2$
12–13	comparison	comparison between prime and prime power Gauß periods
12	Δ_r	the quotient between columns 8 and 3
13	Δ_{c_A}	the quotient between columns 9 and 4; this is the speed-up achieved by substituting the prime Gauß period by the prime power Gauß period.

Comparison of arithmetic between prime power and prime Gauß periods												
	prime power Gauß period					prime Gauß period					comparison	
n	k	r	c_A	#poly	c_Q	k	r	c_A	#poly	c_Q	Δ_r	Δ_{c_A}
171	2	361	0.20	11.24	0.00	12	2053	0.80	45.85	0.00	5.69	4.08
406	2	841	0.48	6.89	0.00	6	2437	1.21	17.34	0.00	2.90	2.52
605	2	1331	0.94	6.72	0.00	6	3631	1.90	13.58	0.00	2.73	2.02
812	1	841	0.69	4.03	0.00	3	2437	1.27	7.40	0.00	2.90	1.84
1014	2	2197	1.74	8.90	0.01	2	2029	0.86	4.37	0.01	0.92	0.49
1210	1	1331	1.26	3.74	0.01	9	10891	11.59	34.34	0.01	8.18	9.19
1378	2	2809	2.23	5.44	0.01	6	8269	6.61	16.09	0.01	2.94	2.96
1711	2	3481	2.82	5.43	0.01	6	10267	10.50	20.22	0.01	2.95	3.72
1830	2	3721	2.98	5.40	0.01	18	32941	59.54	107.90	0.01	8.85	19.98
2028	1	2197	2.29	3.95	0.01	1	2029	1.03	1.77	0.01	0.92	0.45
2211	2	4489	4.29	4.95	0.01	20	44221	77.94	89.81	0.01	9.85	18.16
2485	2	5041	4.98	4.78	0.01	4	9941	10.25	9.83	0.01	1.97	2.06
2500	1	3125	3.65	3.46	0.01	9	22501	34.78	32.99	0.01	7.20	9.53
2756	1	2809	2.95	2.41	0.01	3	8269	6.84	5.58	0.01	2.94	2.32
3081	2	6241	6.44	4.68	0.01	12	36973	62.70	45.53	0.01	5.92	9.74
3249	2	6859	7.57	5.12	0.01	10	32491	48.73	32.96	0.01	4.74	6.44
3403	2	6889	7.12	4.60	0.01	4	13613	15.16	9.80	0.01	1.98	2.13
3660	1	3721	3.95	2.41	0.02	9	32941	59.96	36.64	0.01	8.85	15.20
4422	1	4489	5.43	2.11	0.02	10	44221	79.13	30.80	0.02	9.85	14.58
5050	2	10201	13.22	4.19	0.02	1	5051	4.29	1.36	0.02	0.50	0.32
5253	2	10609	14.32	4.16	0.02	4	21013	32.87	9.55	0.02	1.98	2.29
6250	2	15625	23.67	5.65	0.02	6	37501	64.11	15.29	0.02	2.40	2.71
6806	1	6889	9.06	1.95	0.03	2	13613	15.75	3.40	0.03	1.98	1.74
7203	2	16807	28.32	5.87	0.03	4	28813	46.40	9.62	0.03	1.71	1.64
8515	2	17161	27.61	4.03	0.03	4	34061	61.35	8.95	0.03	1.98	2.22

Table A.15: Times for basic operations (multiplication, squaring) in $\mathbb{F}_{2^n}$ for the test series *PrimepowerGP*.

The times for inversion of the test series *PrimeGP* are listed in Table A.16 and Table A.17. All times are average values for 1000 trials. Table A.16 contains the values for sequential inversion a la Fermat, as described in detail in Section 6.5 for arbitrary normal bases. Table A.17 is a continuation of Table A.16. It compares the sequential inversion to parallel inversion a la Fermat (also described in Section 6.5) and inversion via the EEA, i.e. Algorithm 8.33 adapted to prime Gauß periods.

The key to Table A.16		
column	label	description
1–2	Param.	as columns 1–2 of Table A.14
3–10	in sequential a la Fermat	figures for sequential inversion
3–5	binary	input addition chain for $n-1$ is the binary addition chain
3	t	the average time over 1000 trials in CPU-milliseconds
4	L	the length of the input addition chain for $n-1$
5	vs. $T_{\mathcal{N}}$	the quotient of column 3 of this table and column 3 of Table A.13; it is the speed-up that is achieved by exchanging matrix-based multiplication by polynomial based multiplication for normal prime Gauß periods
6–9	Brauer	input addition chain for $n-1$ is a Brauer addition chain
6–8	t, L, $T_{\mathcal{N}}$	as columns 3–5
9	Δ	the quotient of columns 6 and 3
10–13	power tree	input addition chain for $n-1$ is a power tree addition chain
10–12	t, L, $T_{\mathcal{N}}$	as columns 3–5
13	Δ	the quotient of columns 10 and 3

The key to Table A.17		
column	label	description
1–2	Param.	as columns 1–2 of Table A.14
3–5	a la Fermat	inversion by exponentiation
3	binary	a reprint of column 3 of Table A.16
4–5	in parallel	figures for parallel inversion on two processors, see Algorithm 6.27
4	t	the average time over 1000 trials in CPU-milliseconds
5	Δ	the quotient of columns 3 and 4
6–10	via EEA	inversion by applying the Extended Euclidean Algorithm in $\mathcal{R} = \mathbb{F}_q[x]/(\Phi_{nk+1})$
6–7	in sequential	figures for inversion via EEA
6	t	as column 4
7	Δ	the quotient of columns 3 and 6
8–10	compared to	the speed-up of inversion via EEA in comparison with inversion a la Fermat
8	Brauer	quotient of column 3 with column 6 of Table A.16
9	power tree	quotient of column 3 with column 10 of Table A.16
10	parallel	quotient of columns 3 and 6

Inversion using a normal Gauß period of type (n, k)												
Parameter		in sequential a la Fermat										
n	k	binary			Brauer				power tree			
		t	L	vs. T_N	t	L	vs. T_N	Δ	t	L	vs. T_N	Δ
209	2	1.14	10	15.05	1.14	10	15.39	0.99	1.15	10	15.30	0.99
398	2	3.16	13	27.55	3.15	13	26.64	1.00	2.90	12	26.52	1.09
606	2	6.72	15	34.21	6.72	15	33.51	1.00	5.79	13	33.30	1.16
803	2	8.19	13	41.36	8.14	13	41.46	1.01	8.21	13	41.19	1.00
1018	1	5.98	17	87.92	5.22	15	88.82	1.15	4.88	14	88.74	1.22
1199	2	18.95	16	49.43	18.96	16	49.71	1.00	16.49	14	49.29	1.15
1401	2	23.25	16	54.69	23.17	16	55.49	1.00	20.17	14	55.32	1.15
1601	2	22.05	13	61.61	21.92	13	61.37	1.01	22.02	13	61.03	1.00
1791	2	35.93	19	69.85	31.83	17	70.76	1.13	27.93	15	70.90	1.29
1996	1	15.53	18	137.26	13.64	16	139.29	1.14	12.82	15	138.23	1.21
2212	1	18.66	16	125.67	18.63	16	125.40	1.00	17.44	15	124.68	1.07
2406	2	57.09	17	75.92	56.93	17	75.77	1.00	50.31	15	74.97	1.13
2613	2	60.26	16	80.94	60.00	16	80.86	1.00	56.16	15	81.19	1.07
2802	1	29.06	18	147.26	29.05	18	147.35	1.00	25.75	16	147.10	1.13
3005	2	83.51	19	92.71	78.65	18	93.70	1.06	69.56	16	93.36	1.20
3202	1	28.45	15	159.11	28.22	15	163.30	1.01	28.42	15	162.33	1.00
3401	2	81.63	16	102.93	81.71	16	102.15	1.00	76.08	15	102.40	1.07
3603	2	85.05	16	110.93	85.40	16	109.52	1.00	85.13	16	110.03	1.00
3802	1	41.76	19	204.29	39.50	18	202.48	1.06	37.17	17	203.88	1.12
4002	1	40.64	18	220.16	38.47	17	218.53	1.06	36.00	16	219.44	1.13
4211	2	119.18	17	114.43	120.33	17	113.31	0.99	112.01	16	114.14	1.06
4401	2	128.90	16	108.72	129.81	16	107.51	0.99	128.97	16	108.14	1.00
4602	1	66.76	20	203.27	63.54	19	200.01	1.05	56.37	17	200.72	1.18
4806	2	172.32	18	109.16	172.69	18	109.29	1.00	152.16	16	109.50	1.13
5002	1	68.52	18	214.14	68.92	18	210.51	0.99	60.49	16	211.59	1.13
5199	2	193.64	18	113.61	193.86	18	114.04	1.00	183.40	17	113.41	1.06
5399	2	205.70	18	113.63	204.75	18	116.27	1.00	182.16	16	115.35	1.13
5598	2	248.95	21	121.61	236.59	20	120.79	1.05	199.26	17	120.92	1.25
5812	1	92.48	20	239.55	87.48	19	238.82	1.06	77.90	17	238.43	1.19
6005	2	251.58	20	129.76	239.10	19	130.47	1.05	211.97	17	130.82	1.19
6202	1	88.62	18	254.79	88.90	18	252.84	1.00	83.54	17	253.32	1.06
6396	1	109.22	21	258.59	98.04	19	257.76	1.11	87.34	17	257.40	1.25
6614	2	287.04	20	139.52	289.21	20	138.13	0.99	242.06	17	138.97	1.19
6802	1	99.10	18	271.90	99.67	18	271.35	0.99	93.38	17	272.78	1.06
7005	2	303.13	20	148.44	287.93	19	147.49	1.05	255.78	17	147.42	1.19
7205	2	260.15	17	153.75	260.42	17	153.03	1.00	260.42	17	153.59	1.00
7410	1	118.14	20	303.50	105.97	18	302.55	1.11	105.86	18	303.54	1.12
7602	1	120.28	20	313.48	113.87	19	313.95	1.06	108.03	18	312.66	1.11
7803	2	342.40	21	171.51	325.67	20	170.33	1.05	273.86	17	170.66	1.25
8003	2	313.66	19	176.93	295.36	18	177.20	1.06	278.83	17	176.44	1.12
8218	1	112.75	17	330.06	111.84	17	331.87	1.01	112.89	17	328.58	1.00
8411	2	386.15	19	158.54	385.66	19	158.48	1.00	364.93	18	158.21	1.06
8601	2	396.18	18	154.18	372.37	17	152.51	1.06	372.19	17	152.67	1.06
8802	1	155.62	18	289.68	155.05	18	292.04	1.00	156.38	18	289.34	1.00
9006	2	493.32	20	150.85	496.83	20	149.05	0.99	441.57	18	149.79	1.12
9202	1	194.38	21	299.83	184.67	20	299.16	1.05	164.99	18	299.52	1.18
9396	1	195.92	20	291.21	195.65	20	294.44	1.00	175.39	18	293.82	1.12
9603	2	499.59	18	151.29	500.67	18	150.38	1.00	470.06	17	150.52	1.06
9802	1	197.54	19	299.44	197.38	19	300.72	1.00	175.72	17	300.45	1.12
9998	2	585.14	20	155.33	585.47	20	155.61	1.00	491.49	17	155.98	1.19

Table A.16: Times for inversion in $\mathbb{F}_{2^n}$ for the test series *PrimeGP*. This table contains the figures for sequential inversion a la Fermat.

Inversion using a normal Gauß period of type (n, k)									
Parameter		a la Fermat			via EEA				
n	k	binary	in parallel		in sequential		compared to		
		t	t	Δ	t	Δ	Brauer	power tree	parallel
209	2	1.14	1.21	0.94	0.54	2.11	2.12	2.13	2.24
398	2	3.16	2.64	1.19	1.38	2.28	2.28	2.10	1.91
606	2	6.72	5.16	1.30	2.69	2.50	2.50	2.15	1.92
803	2	8.19	7.10	1.15	4.36	1.88	1.87	1.88	1.63
1018	1	5.98	4.18	1.43	2.06	2.90	2.53	2.37	2.03
1199	2	18.95	14.41	1.32	8.96	2.12	2.12	1.84	1.61
1401	2	23.25	17.49	1.33	12.01	1.94	1.93	1.68	1.46
1601	2	22.05	20.45	1.08	15.20	1.45	1.44	1.45	1.35
1791	2	35.93	22.68	1.58	18.77	1.91	1.70	1.49	1.21
1996	1	15.53	10.67	1.46	6.52	2.38	2.09	1.97	1.64
2212	1	18.66	15.34	1.22	7.83	2.38	2.38	2.23	1.96
2406	2	57.09	43.92	1.30	32.97	1.73	1.73	1.53	1.33
2613	2	60.26	49.21	1.22	38.75	1.56	1.55	1.45	1.27
2802	1	29.06	21.50	1.35	12.04	2.41	2.41	2.14	1.79
3005	2	83.51	56.32	1.48	50.34	1.66	1.56	1.38	1.12
3202	1	28.45	24.53	1.16	15.41	1.85	1.83	1.84	1.59
3401	2	81.63	66.08	1.24	63.85	1.28	1.28	1.19	1.03
3603	2	85.05	68.96	1.23	71.12	1.20	1.20	1.20	0.97
3802	1	41.76	28.51	1.46	21.19	1.97	1.86	1.75	1.35
4002	1	40.64	29.55	1.38	23.36	1.74	1.65	1.54	1.26
4211	2	119.18	98.14	1.21	96.63	1.23	1.25	1.16	1.02
4401	2	128.90	112.38	1.15	106.38	1.21	1.22	1.21	1.06
4602	1	66.76	46.27	1.44	30.60	2.18	2.08	1.84	1.51
4806	2	172.32	132.28	1.30	125.45	1.37	1.38	1.21	1.05
5002	1	68.52	52.88	1.30	35.74	1.92	1.93	1.69	1.48
5199	2	193.64	148.30	1.31	146.15	1.33	1.33	1.25	1.01
5399	2	205.70	157.18	1.31	156.81	1.31	1.31	1.16	1.00
5598	2	248.95	163.09	1.53	168.46	1.48	1.40	1.18	0.97
5812	1	92.48	64.01	1.44	47.72	1.94	1.83	1.63	1.34
6005	2	251.58	173.72	1.45	193.13	1.30	1.24	1.10	0.90
6202	1	88.62	68.38	1.30	53.94	1.64	1.65	1.55	1.27
6396	1	109.22	71.75	1.52	57.53	1.90	1.70	1.52	1.25
6614	2	287.04	198.09	1.45	235.32	1.22	1.23	1.03	0.84
6802	1	99.10	76.41	1.30	64.45	1.54	1.55	1.45	1.19
7005	2	303.13	208.90	1.45	264.76	1.14	1.09	0.97	0.79
7205	2	260.15	212.33	1.23	276.56	0.94	0.94	0.94	0.77
7410	1	118.14	82.41	1.43	75.69	1.56	1.40	1.40	1.09
7602	1	120.28	83.20	1.45	79.30	1.52	1.44	1.36	1.05
7803	2	342.40	224.79	1.52	327.32	1.05	0.99	0.84	0.69
8003	2	313.66	227.88	1.38	340.69	0.92	0.87	0.82	0.67
8218	1	112.75	98.31	1.15	92.49	1.22	1.21	1.22	1.06
8411	2	386.15	300.31	1.29	376.57	1.03	1.02	0.97	0.80
8601	2	396.18	326.12	1.21	394.79	1.00	0.94	0.94	0.83
8802	1	155.62	128.43	1.21	106.42	1.46	1.46	1.47	1.21
9006	2	493.32	365.88	1.35	429.82	1.15	1.16	1.03	0.85
9202	1	194.38	137.57	1.41	115.77	1.68	1.60	1.43	1.19
9396	1	195.92	144.99	1.35	120.23	1.63	1.63	1.46	1.21
9603	2	499.59	412.95	1.21	491.26	1.02	1.02	0.96	0.84
9802	1	197.54	154.94	1.27	131.39	1.50	1.50	1.34	1.18
9998	2	585.14	431.94	1.35	527.09	1.11	1.11	0.93	0.82

Table A.17: Times for inversion in $\mathbb{F}_{2^n}$ for the test series *PrimeGP*. This table contains the figures for parallel inversion a la Fermat and the inversion with the help of the EEA.

Table A.18 documents the experiments on multiplication and squaring for the test series *squarefreeGP*. Its goal is to give a comparison between prime Gauß periods and square-free Gauß periods as described in Section 9.2. All figures are the average of 10000 trials. The table is similar to Table A.15.

The key to Table A.18		
column	label	description
1	n	the degree of the extension field $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$ for which normal prime and square-free Gauß periods are compared
2–6	square-free Gauß period	parameters and times for square-free Gauß periods of type $(n, \mathcal{K})$ over $\mathbb{F}_2$
2–6	k	the order of the subgroup $\mathcal{K}$
3	r	$\mathcal{K} \subseteq \mathbb{Z}_r^\times$ and $\phi(r) = nk$
4	c_A	the average time over 10000 multiplications in CPU-milliseconds
5	#poly	the number of multiplications of randomly chosen polynomials in $\mathbb{F}_2[x]$ of degree less than n that can be computed in the same time
6	c_Q	the average time over 10000 squarings in CPU-milliseconds
7–11	prime Gauß period	as columns 2–6 but for prime Gauß period of type (n, k) over $\mathbb{F}_2$
12–13	comparison	comparison between prime and square-free Gauß periods
12	Δ_r	the quotient between columns 8 and 3
13	Δ_{c_A}	the quotient between columns 9 and 4; this is the speed-up achieved by substituting the prime Gauß period by the square-free Gauß period.

Comparison of arithmetic between square-free and prime Gauß periods												
	square free Gauß period					prime Gauß period					comparison	
n	k	r	c_A	#poly	c_Q	k	r	c_A	#poly	c_Q	Δ_r	Δ_{c_A}
198	2	437	0.13	4.44	0.00	22	4357	2.72	96.57	0.00	9.97	21.73
396	2	851	0.29	3.67	0.00	11	4357	2.75	35.43	0.00	5.12	9.66
598	2	1797	0.71	4.62	0.00	15	8971	8.45	54.80	0.00	4.99	11.86
805	4	3337	1.81	9.67	0.00	6	4831	3.37	17.98	0.00	1.45	1.86
1004	2	2515	1.32	6.22	0.01	5	5021	3.56	16.77	0.01	2.00	2.70
1219	4	5029	3.57	9.62	0.01	4	4877	3.50	9.40	0.01	0.97	0.98
1398	2	3269	1.86	4.34	0.01	2	2797	1.54	3.58	0.01	0.86	0.83
1602	2	3401	1.95	3.80	0.01	6	9613	9.79	19.07	0.01	2.83	5.02
1790	2	3949	2.17	3.96	0.01	2	3581	1.98	3.63	0.01	0.91	0.91
1996	2	5991	4.53	7.63	0.01	1	1997	0.90	1.51	0.01	0.33	0.20
2206	2	6621	5.20	6.05	0.01	3	6619	5.14	5.98	0.01	1.00	0.99
2398	2	7197	5.53	5.39	0.01	6	14389	15.65	15.25	0.01	2.00	2.83
2604	2	6515	5.19	4.50	0.01	7	18229	25.67	22.25	0.01	2.80	4.95
2796	2	6071	4.71	3.77	0.01	1	2797	1.69	1.35	0.01	0.46	0.36
2998	2	8997	8.84	6.68	0.01	6	17989	25.27	19.11	0.01	2.00	2.86
3212	2	6739	5.46	3.67	0.01	9	28909	46.47	31.22	0.01	4.29	8.51
3406	2	10221	10.61	6.68	0.01	10	34061	74.16	46.68	0.01	3.33	6.99
3597	4	16793	20.80	12.57	0.01	4	14389	15.75	9.52	0.01	0.86	0.76
3799	4	15517	16.81	9.86	0.01	10	37991	77.58	45.49	0.01	2.45	4.62
4002	2	8201	6.63	3.75	0.02	1	4003	2.37	1.34	0.02	0.49	0.36
4202	4	26427	42.97	19.62	0.02	5	21011	32.84	14.99	0.02	0.80	0.76
4410	2	9329	9.75	3.82	0.02	2	8821	8.57	3.36	0.02	0.95	0.88
4602	2	13809	15.54	5.69	0.02	1	4603	3.46	1.27	0.02	0.33	0.22
4807	4	19693	30.05	9.91	0.02	6	28843	46.36	15.28	0.02	1.46	1.54
5004	2	10583	11.89	3.75	0.02	3	15013	16.50	5.20	0.02	1.42	1.39
5198	2	10669	12.03	3.50	0.02	6	31189	48.67	14.17	0.02	2.92	4.05
5395	4	21877	35.36	9.73	0.02	6	32371	49.57	13.64	0.02	1.48	1.40
5604	2	14015	15.94	4.28	0.02	7	39229	78.31	21.04	0.02	2.80	4.91
5804	2	14515	16.38	4.20	0.02	5	29021	46.59	11.94	0.02	2.00	2.84
5996	2	14995	16.86	4.24	0.02	3	17989	25.69	6.47	0.02	1.20	1.52
6198	2	18597	27.79	6.67	0.02	6	37189	77.55	18.63	0.02	2.00	2.79
6404	2	16015	17.45	3.96	0.02	3	19213	28.95	6.56	0.02	1.20	1.66
6604	4	26977	45.23	9.99	0.02	3	19813	30.09	6.65	0.02	0.73	0.67
6806	2	13861	16.06	3.40	0.02	2	13613	15.52	3.28	0.02	0.98	0.97
6996	2	16331	17.64	3.69	0.03	5	34981	74.80	15.67	0.03	2.14	4.24
7206	2	21621	35.58	7.31	0.03	6	43237	96.48	19.83	0.03	2.00	2.71
7380	2	15023	17.23	3.44	0.03	5	36901	77.73	15.53	0.03	2.46	4.51
7598	2	15517	17.33	3.42	0.03	6	45589	99.70	19.67	0.03	2.94	5.75
7794	4	32927	71.10	13.72	0.03	5	38971	78.33	15.12	0.03	1.18	1.10
7980	4	38773	74.30	14.32	0.03	5	39901	78.65	15.16	0.03	1.03	1.06
8199	4	34637	71.50	12.75	0.03	4	32797	74.61	13.30	0.03	0.95	1.04
8395	4	36949	73.63	11.29	0.03	4	33581	74.98	11.49	0.03	0.91	1.02
8622	2	25869	43.07	6.03	0.03	3	25867	42.86	6.00	0.03	1.00	1.00
8782	2	26349	43.78	5.77	0.04	3	26347	43.77	5.76	0.03	1.00	1.00
9012	2	22535	36.71	4.62	0.04	5	45061	97.06	12.23	0.03	2.00	2.64
9204	2	23015	37.58	4.63	0.03	5	46021	97.36	12.00	0.03	2.00	2.59
9379	4	37909	74.19	8.50	0.03	4	37517	78.16	8.96	0.03	0.99	1.05
9596	2	23995	38.69	4.29	0.04	3	28789	46.72	5.18	0.03	1.20	1.21
9798	4	40091	75.02	8.07	0.04	2	19597	30.08	3.24	0.04	0.49	0.40
9996	2	20291	31.55	3.34	0.04	3	29989	48.25	5.10	0.04	1.48	1.53

Table A.18: Times for basic operations (multiplication, squaring) in $\mathbb{F}_{2^n}$ for the test series *squarefreeGP*.

A.3. Parallel exponentiation. The times for Figure 10.1 are documented Table A.19. The experiments used the test series *Arbitrary*, i.e. the finite field $\mathbb{F}_{2^n}$ was represented by a polynomial basis with an arbitrary irreducible polynomial as modulus. But here, the squaring operation was substituted by a multiplication to guarantee that multiplication and squaring have nearly the same cost, i.e. $c_Q = c_A$. The times are the average of 10 trials for each n.

The key to Table A.19		
column	label	description
1	deg	the degree of the extension field of $\mathbb{F}_2$
2–4	sequential	sequential exponentiation in $\mathbb{F}_{2^n}$
2	binary/sec	the average times for 10 trials in CPU-milliseconds if the binary addition chain is applied to exponentiation
3–4	Brauer	exponentiation is done in sequential using a Brauer addition chain (Algorithm 3.12)
3	sec	the average times for 10 trials in CPU-milliseconds
4	spd	quotient of columns 2 and 3; this can be regarded as the speed-up of Brauer addition chains in comparison with the binary addition chain
5–10	parallel	figures for different parallel exponentiation algorithms
5–6	Brauer (2 proc)	figures of the parallel version of Brauer's algorithm described in Section 10 on 2 processors
5, 6	sec, spd	similar to columns 3–4
7–8	Brauer (8 proc)	figures of the parallel version of Brauer's algorithm described in Section 10 on 8 processors
7, 8	sec, spd	similar to columns 3–4
9–10	Borodin & Munro	figures of Algorithm 4.3 which runs on 2 processors
7, 8	sec, spd	similar to columns 3–4

Parallel exponentiation if only multiplications are used									
deg	sequential			parallel					
n	binary	Brauer		Brauer (2 procs)		Brauer (8 procs)		Borodin & Munro	
	sec	sec	spd	sec	spd	sec	spd	sec	spd
209	0.02	0.01	1.12	0.01	1.39	0.01	1.42	0.02	1.00
398	0.09	0.08	1.14	0.06	1.44	0.06	1.42	0.07	1.27
606	0.25	0.21	1.17	0.17	1.43	0.17	1.45	0.19	1.29
803	0.50	0.41	1.21	0.34	1.45	0.33	1.49	0.35	1.41
1018	0.83	0.71	1.17	0.57	1.45	0.56	1.49	0.58	1.43
1199	1.44	1.21	1.19	0.99	1.45	0.97	1.49	0.99	1.46
1401	2.14	1.77	1.21	1.46	1.46	1.43	1.50	1.44	1.48
1601	3.03	2.49	1.22	2.07	1.46	2.01	1.51	2.03	1.49
1791	3.92	3.22	1.22	2.69	1.46	2.62	1.50	2.64	1.49
1996	5.16	4.20	1.23	3.53	1.46	3.43	1.50	3.46	1.49
2212	7.47	6.04	1.24	5.08	1.47	4.94	1.51	4.98	1.50
2406	9.39	7.57	1.24	6.42	1.46	6.24	1.50	6.26	1.50
2613	11.73	9.43	1.24	7.99	1.47	7.78	1.51	7.80	1.50
2802	14.06	11.25	1.25	9.54	1.47	9.33	1.51	9.34	1.50
3005	16.70	13.77	1.21	11.38	1.47	11.05	1.51	11.11	1.50
3202	19.90	16.37	1.22	13.56	1.47	13.21	1.51	13.22	1.51
3401	23.39	19.05	1.23	15.85	1.48	15.46	1.51	15.49	1.51
3603	26.97	21.87	1.23	18.27	1.48	17.83	1.51	17.85	1.51
3802	30.70	24.82	1.24	20.88	1.47	20.26	1.51	20.26	1.51
4002	35.06	28.22	1.24	23.75	1.48	23.13	1.52	23.12	1.52
4211	39.77	32.00	1.24	26.95	1.48	26.29	1.51	26.21	1.52
4401	48.05	38.45	1.25	32.53	1.48	31.68	1.52	31.58	1.52
4602	54.46	43.67	1.25	36.88	1.48	35.99	1.51	35.94	1.52
4806	61.85	49.51	1.25	41.90	1.48	40.92	1.51	40.76	1.52
5002	67.55	53.90	1.25	45.83	1.47	44.71	1.51	44.67	1.51
5199	75.66	59.88	1.26	50.91	1.49	49.78	1.52	49.63	1.52
5399	83.59	66.27	1.26	56.44	1.48	55.12	1.52	54.90	1.52
5598	88.58	70.23	1.26	59.93	1.48	58.60	1.51	58.31	1.52
5812	96.77	76.59	1.26	65.48	1.48	63.91	1.51	63.57	1.52
6005	103.49	81.81	1.27	69.94	1.48	68.46	1.51	68.25	1.52
6202	110.30	86.71	1.27	74.36	1.48	72.59	1.52	72.49	1.52
6396	121.30	95.09	1.28	81.57	1.49	79.80	1.52	79.26	1.53
6614	128.77	100.90	1.28	86.76	1.48	84.84	1.52	84.57	1.52
6802	132.91	103.78	1.28	89.13	1.49	87.17	1.52	86.49	1.54
7005	142.42	111.70	1.28	96.17	1.48	94.07	1.51	93.73	1.52
7205	150.94	118.28	1.28	101.86	1.48	99.65	1.51	99.22	1.52
7410	159.44	124.31	1.28	107.50	1.48	105.02	1.52	104.53	1.53
7602	164.87	128.87	1.28	111.14	1.48	108.94	1.51	108.80	1.52
7803	172.00	138.61	1.24	115.94	1.48	113.70	1.51	113.18	1.52
8003	177.78	143.10	1.24	119.69	1.49	117.22	1.52	116.48	1.53
8218	198.09	159.53	1.24	133.28	1.49	130.65	1.52	129.86	1.53
8411	232.27	186.60	1.24	156.60	1.48	153.56	1.51	152.39	1.52
8601	259.11	207.39	1.25	174.62	1.48	170.93	1.52	169.82	1.53
8802	288.63	230.67	1.25	194.52	1.48	190.48	1.52	189.45	1.52
9006	309.57	246.89	1.25	208.29	1.49	204.45	1.51	203.20	1.52
9202	328.45	261.13	1.26	220.72	1.49	216.23	1.52	214.83	1.53
9396	351.22	279.01	1.26	235.99	1.49	231.42	1.52	229.54	1.53
9603	371.28	294.91	1.26	249.42	1.49	244.59	1.52	243.09	1.53
9802	392.73	311.48	1.26	263.99	1.49	258.92	1.52	257.63	1.52
9998	406.50	321.63	1.26	272.90	1.49	267.19	1.52	266.34	1.53

Table A.19: Comparison of sequential and parallel exponentiation algorithms if multiplications and squaring have same cost.

Tables A.20–A.23 list the results of the experiments described in Section 10.4. All four tables have the same structure. They contain average times and speed-ups for the four test series *Arbitrary*, *Sparse*, *Sedimentary* and *PrimeGP*. For the latter three test series all values are the average of 100 trials. For the test series *Arbitrary* the experiments were repeated only 10 times for each n. The times are given in CPU-seconds, the speed-up is with respect to exponentiation based on weighted 2-Brauer addition chains.

The key to Tables A.20–A.23		
column	label	description
1–3	Parameter	characterization of the basis representation
1	n	degree of the extension field over $\mathbb{F}_2$
2	$—/e_3, e_2, e_1/k$	*Table A.20:* no such column, *Table A.21:* indices of the non-negative coefficients in the modulus: only e_1 given: $f = x^n + x^{e_1} + 1 \in \mathbb{F}_2[x]$ is a trinomial. Else $f = x^n + x^{e_3} + x^{e_2} + x^{e_1} + 1 \in \mathbb{F}_2[x]$. *Table A.22:* k is the degree of the sediment h and $f = x^n + h \in \mathbb{F}_2[x]$. *Table A.23:* k is the order of the subgroup $\mathcal{K} \subseteq \mathbb{Z}_{nk+1}^{\times}$.
3	c	the ratio $\frac{c_Q}{c_A}$
4	seq.	average time in CPU-seconds for 10 (Table A.20) or 100 trials (all other tables), respectively, for sequential exponentiation with respect to a weighted 2-Brauer addition chain
5–14	parallel	figures for the implementation of the parallel exponentiation algorithm described in Section 10
5	2	the average times in CPU-seconds for the parallel algorithm performed on 2 processors
6	spd.	the speed-up compared to the sequential algorithm, i.e. the quotient of columns 4 and 5
7, 8	4, spd.	times and speed-up similar to columns 5–6 but for 4 processors
9, 10	8, spd.	times and speed-up similar to columns 5–6 but for 8 processors
11, 12	16, spd.	times and speed-up similar to columns 5–6 but for 16 processors
13, 14	32, spd.	times and speed-up similar to columns 5–6 but for 32 processors

The closing Table A.24 compares the predicted speed-up (see Section 4.3) to the achieved speed-up (see Section 10.4) for the four test series *Arbitrary*, *Sparse*, *Sedimentary* and *PrimeGP*.

The key to Tables A.20–A.23		
column	label	description
1	Param./n	the degree of the field extension
2–4	*Arbitrary*	figures for the test series *Arbitrary*
2	exp.	the maximal speed-up in relation to repeated squaring that is given by experiment; the maximum is taken over the average times for 2, 4, 8, 16 and 32 processors
3	theo.	the expected speed-up as documented in Table A.6, column 10.
4	%	the quotient of columns 3 and 4
5–7	*Sparse*	figures for the test series *Sparse*
5	exp.	as column 2
6	theo.	the expected speed-up as documented in Table A.7, column 10
7	%	the quotient of columns 5 and 6
8–10	*Sedimentary*	figures for the test series *Sedimentary*
8	exp.	as column 2
9	theo.	the expected speed-up as documented in Table A.8, column 10
10	%	the quotient of columns 8 and 9
11–13	*PrimeGP*	figures for the test series *PrimeGP*
11	exp.	as column 2
12	theo.	the optimal depth in case of weight $(0, 1)$ is given by Fact 4.15
13	%	the quotient of columns 11 and 12

Parallel exponentiation in $\mathbb{F}_2[x]/(f)$ with f arbitrary												
Parameter		seq.	parallel (number of processors)									
n	c		2	spd.	4	spd.	8	spd.	16	spd.	32	spd.
209	0.74	0.01	0.01	1.08	0.01	1.32	0.01	1.34	0.01	1.29	0.01	1.29
398	0.64	0.05	0.04	1.23	0.04	1.31	0.04	1.34	0.04	1.22	0.04	1.35
606	0.60	0.14	0.11	1.23	0.10	1.33	0.10	1.34	0.11	1.27	0.10	1.34
803	0.64	0.30	0.24	1.23	0.23	1.31	0.22	1.33	0.23	1.29	0.23	1.28
1018	0.69	0.52	0.43	1.20	0.40	1.28	0.40	1.29	0.41	1.27	0.42	1.24
1199	0.62	0.82	0.68	1.21	0.63	1.30	0.62	1.31	0.64	1.29	0.65	1.27
1401	0.63	1.22	1.01	1.20	0.94	1.29	0.94	1.30	0.94	1.29	0.96	1.26
1601	0.64	1.75	1.47	1.19	1.37	1.28	1.36	1.28	1.37	1.28	1.40	1.25
1791	0.67	2.35	1.96	1.20	1.84	1.28	1.84	1.28	1.86	1.27	1.88	1.25
1996	0.70	3.15	2.66	1.18	2.53	1.24	2.51	1.25	2.58	1.22	2.57	1.23
2212	0.64	4.27	3.61	1.18	3.37	1.26	3.36	1.27	3.37	1.27	3.43	1.24
2406	0.64	5.35	4.52	1.18	4.25	1.26	4.22	1.27	4.23	1.27	4.30	1.24
2613	0.65	6.68	5.68	1.18	5.33	1.25	5.29	1.26	5.30	1.26	5.41	1.24
2802	0.65	8.04	6.87	1.17	6.46	1.24	6.40	1.26	6.40	1.26	6.56	1.23
3005	0.67	9.75	8.32	1.17	7.82	1.25	7.79	1.25	7.78	1.25	7.97	1.22
3202	0.67	11.62	9.94	1.17	9.34	1.24	9.31	1.25	9.32	1.25	9.54	1.22
3401	0.68	13.73	11.80	1.16	11.12	1.23	11.08	1.24	11.08	1.24	11.32	1.21
3603	0.70	16.04	13.81	1.16	13.07	1.23	12.98	1.24	13.01	1.23	13.32	1.20
3802	0.71	18.47	15.93	1.16	15.14	1.22	15.05	1.23	15.08	1.22	15.44	1.20
4002	0.73	21.46	18.55	1.16	17.64	1.22	17.57	1.22	17.57	1.22	18.00	1.19
4211	0.68	23.07	19.47	1.19	18.78	1.23	18.76	1.23	18.82	1.23	19.25	1.20
4401	0.67	27.56	23.26	1.18	22.47	1.23	22.49	1.23	22.70	1.21	22.94	1.20
4602	0.68	31.40	26.47	1.19	25.64	1.22	25.49	1.23	25.55	1.23	26.05	1.21
4806	0.67	35.31	29.86	1.18	28.76	1.23	28.72	1.23	28.75	1.23	29.29	1.21
5002	0.67	38.58	32.56	1.18	31.45	1.23	31.29	1.23	31.41	1.23	32.05	1.20
5199	0.66	42.81	36.02	1.19	34.85	1.23	34.70	1.23	34.73	1.23	35.22	1.22
5399	0.67	47.44	40.08	1.18	38.81	1.22	38.55	1.23	38.71	1.23	39.34	1.21
5598	0.66	50.25	42.31	1.19	41.10	1.22	40.99	1.23	40.86	1.23	41.50	1.21
5812	0.67	55.14	46.50	1.19	45.22	1.22	44.81	1.23	44.88	1.23	45.77	1.20
6005	0.67	59.29	50.25	1.18	48.81	1.21	48.47	1.22	48.41	1.22	49.51	1.20
6202	0.65	62.34	52.51	1.19	51.13	1.22	50.94	1.22	50.86	1.23	51.47	1.21
6396	0.67	68.94	58.25	1.18	56.70	1.22	56.57	1.22	56.72	1.22	57.81	1.19
6614	0.67	72.99	61.79	1.18	60.07	1.22	59.85	1.22	59.90	1.22	61.06	1.20
6802	0.64	74.29	62.63	1.19	60.72	1.22	60.42	1.23	60.37	1.23	60.90	1.22
7005	0.67	80.84	68.44	1.18	66.53	1.22	66.33	1.22	66.33	1.22	67.91	1.19
7205	0.67	85.92	72.92	1.18	70.96	1.21	70.60	1.22	70.80	1.21	72.37	1.19
7410	0.67	90.66	77.01	1.18	74.82	1.21	74.45	1.22	74.46	1.22	76.69	1.18
7602	0.67	93.74	79.83	1.17	77.33	1.21	77.18	1.21	77.52	1.21	79.27	1.18
7803	0.67	97.95	83.42	1.17	81.01	1.21	80.73	1.21	80.59	1.22	83.22	1.18
8003	0.67	101.10	85.84	1.18	83.75	1.21	83.43	1.21	83.08	1.22	85.70	1.18
8218	0.66	112.40	95.68	1.17	93.26	1.21	92.67	1.21	92.43	1.22	94.64	1.19
8411	0.66	131.15	111.38	1.18	108.21	1.21	107.84	1.22	107.69	1.22	110.39	1.19
8601	0.66	145.37	123.60	1.18	120.05	1.21	119.93	1.21	119.57	1.22	122.12	1.19
8802	0.67	162.07	138.19	1.17	134.48	1.21	133.64	1.21	133.68	1.21	137.29	1.18
9006	0.67	173.42	148.37	1.17	144.23	1.20	143.33	1.21	142.84	1.21	146.83	1.18
9202	0.67	184.20	157.23	1.17	152.85	1.21	153.10	1.20	152.57	1.21	156.31	1.18
9396	0.66	195.52	166.39	1.18	162.18	1.21	161.61	1.21	161.16	1.21	164.70	1.19
9603	0.66	205.97	175.67	1.17	170.52	1.21	170.64	1.21	169.89	1.21	173.57	1.19
9802	0.66	218.67	186.57	1.17	181.81	1.20	181.73	1.20	180.69	1.21	185.38	1.18
9998	0.66	225.50	192.42	1.17	186.66	1.21	186.44	1.21	186.02	1.21	190.34	1.18

Table A.20: Times for exponentiation in $\mathbb{F}_{2^n}$ for the test series *Arbitrary*.

Parallel exponentiation in $\mathbb{F}_2[x]/(f)$ with f sparse													
Parameter			seq.	parallel (number of processors)									
n	e_3, e_2, e_1	c		2	spd.	4	spd.	8	spd.	16	spd.	32	spd.
209	6	0.47	0.00	0.00	1.21	0.00	1.40	0.00	1.43	0.00	1.31	0.00	1.38
398	7, 6, 2	0.29	0.02	0.01	1.35	0.01	1.47	0.01	1.65	0.01	1.65	0.01	1.62
606	165	0.18	0.03	0.02	1.50	0.02	1.76	0.02	1.91	0.02	2.04	0.02	1.57
803	14, 9, 2	0.18	0.06	0.04	1.48	0.03	1.84	0.03	1.84	0.03	1.98	0.04	1.66
1018	12, 10, 5	0.19	0.09	0.06	1.46	0.05	1.79	0.05	1.80	0.04	1.93	0.05	1.61
1199	114	0.11	0.13	0.09	1.53	0.06	2.15	0.06	2.34	0.05	2.40	0.06	2.11
1401	92	0.11	0.18	0.11	1.58	0.08	2.21	0.07	2.45	0.07	2.49	0.08	2.27
1601	548	0.10	0.24	0.15	1.62	0.11	2.24	0.09	2.53	0.09	2.52	0.10	2.39
1791	190	0.10	0.28	0.18	1.59	0.13	2.21	0.11	2.49	0.11	2.47	0.12	2.34
1996	307	0.11	0.34	0.21	1.58	0.16	2.15	0.14	2.43	0.14	2.40	0.14	2.33
2212	423	0.08	0.50	0.30	1.64	0.21	2.40	0.17	2.84	0.17	2.84	0.17	2.87
2406	14, 12, 5	0.08	0.65	0.40	1.63	0.28	2.34	0.24	2.76	0.24	2.75	0.23	2.78
2613	14, 12, 9	0.08	0.78	0.48	1.63	0.33	2.36	0.28	2.78	0.28	2.82	0.27	2.85
2802	18, 14, 3	0.08	0.88	0.55	1.61	0.37	2.36	0.31	2.80	0.31	2.88	0.30	2.89
3005	14, 12, 5	0.08	1.01	0.63	1.60	0.43	2.34	0.36	2.76	0.36	2.80	0.35	2.85
3202	24, 12, 3	0.07	1.18	0.73	1.61	0.50	2.37	0.41	2.83	0.40	2.93	0.40	2.94
3401	531	0.06	1.23	0.76	1.62	0.50	2.46	0.40	3.05	0.38	3.21	0.38	3.25
3603	23, 11, 6	0.08	1.44	0.91	1.59	0.60	2.38	0.52	2.79	0.50	2.88	0.50	2.90
3802	16, 15, 8	0.08	1.58	0.97	1.62	0.66	2.38	0.57	2.77	0.55	2.85	0.55	2.89
4002	24, 11, 6	0.08	1.71	1.04	1.65	0.73	2.35	0.63	2.72	0.61	2.80	0.60	2.84
4211	14, 12, 6	0.07	2.12	1.27	1.67	0.86	2.46	0.71	2.98	0.69	3.05	0.67	3.14
4401	394	0.05	2.36	1.36	1.73	0.88	2.69	0.68	3.46	0.67	3.53	0.63	3.76
4602	675	0.05	2.58	1.50	1.72	0.96	2.69	0.74	3.47	0.68	3.79	0.67	3.87
4806	2349	0.05	2.93	1.70	1.73	1.07	2.74	0.83	3.54	0.76	3.84	0.73	4.02
5002	39, 23, 5	0.05	3.35	1.97	1.70	1.28	2.61	1.02	3.28	0.99	3.40	0.94	3.55
5199	1546	0.04	3.50	2.03	1.73	1.27	2.76	0.97	3.62	0.87	4.03	0.85	4.13
5399	485	0.04	3.81	2.21	1.72	1.38	2.76	1.05	3.64	0.97	3.94	0.92	4.15
5598	101	0.04	4.07	2.36	1.72	1.48	2.75	1.12	3.63	1.00	4.05	0.99	4.11
5812	295	0.04	4.37	2.55	1.72	1.60	2.74	1.20	3.62	1.08	4.03	1.07	4.08
6005	28, 12, 2	0.05	4.84	2.87	1.69	1.86	2.60	1.47	3.30	1.40	3.46	1.34	3.61
6202	867	0.04	4.95	2.90	1.71	1.81	2.74	1.37	3.62	1.20	4.11	1.21	4.09
6396	91	0.04	5.32	3.10	1.71	1.94	2.74	1.46	3.65	1.33	4.00	1.28	4.16
6614	2105	0.04	5.70	3.33	1.71	2.07	2.75	1.56	3.66	1.39	4.11	1.37	4.16
6802	29, 25, 3	0.05	6.32	3.77	1.68	2.42	2.62	1.89	3.34	1.79	3.52	1.73	3.64
7005	14, 11, 4	0.05	6.65	3.97	1.68	2.55	2.61	2.00	3.33	1.90	3.51	1.85	3.60
7205	21, 5, 2	0.05	6.93	4.13	1.68	2.64	2.62	2.06	3.36	1.97	3.51	1.94	3.58
7410	2179	0.04	6.92	4.09	1.69	2.55	2.71	1.93	3.58	1.75	3.96	1.84	3.75
7602	555	0.04	7.22	4.27	1.69	2.67	2.70	2.03	3.56	1.88	3.85	1.96	3.68
7803	19, 14, 2	0.05	7.93	4.77	1.66	3.08	2.58	2.42	3.28	2.27	3.49	2.24	3.55
8003	26, 21, 2	0.05	8.25	5.00	1.65	3.23	2.56	2.56	3.22	2.44	3.38	2.43	3.39
8218	1443	0.04	8.56	5.07	1.69	3.15	2.72	2.38	3.60	2.11	4.07	2.12	4.04
8411	30, 27, 5	0.04	10.26	6.12	1.68	3.82	2.68	2.87	3.58	2.63	3.90	2.87	3.57
8601	734	0.03	10.76	6.30	1.71	3.81	2.82	2.76	3.90	2.37	4.55	2.31	4.65
8802	2139	0.03	11.71	6.85	1.71	4.11	2.85	2.91	4.02	2.55	4.59	2.42	4.85
9006	1477	0.03	12.45	7.27	1.71	4.35	2.86	3.05	4.08	2.58	4.82	2.48	5.02
9202	211	0.03	12.99	7.60	1.71	4.54	2.86	3.15	4.13	2.68	4.85	2.59	5.02
9396	369	0.03	14.00	8.19	1.71	4.86	2.88	3.35	4.18	2.82	4.96	2.76	5.07
9603	19, 10, 4	0.04	15.41	9.11	1.69	5.55	2.77	3.96	3.89	3.47	4.44	3.46	4.45
9802	12, 10, 3	0.03	16.02	9.47	1.69	5.69	2.82	4.06	3.95	3.51	4.57	3.67	4.36
9998	4013	0.03	16.16	9.38	1.72	5.42	2.98	3.80	4.25	3.19	5.06	3.37	4.79

Table A.21: Times for exponentiation in $\mathbb{F}_{2^n}$ for the test series *Sparse*.

Parallel exponentiation in $\mathbb{F}_2[x]/(f)$ with $f = x^n + h$ sedimentary													
Parameter			seq.	parallel (number of processors)									
n	k	c		2	spd.	4	spd.	8	spd.	16	spd.	32	spd.
209	5	0.52	0.01	0.00	1.21	0.00	1.36	0.00	1.39	0.00	1.34	0.00	1.34
398	7	0.32	0.02	0.01	1.36	0.01	1.50	0.01	1.63	0.01	1.63	0.01	1.65
606	9	0.25	0.04	0.03	1.32	0.02	1.74	0.02	1.79	0.02	1.85	0.03	1.52
803	8	0.24	0.07	0.05	1.43	0.04	1.70	0.04	1.73	0.04	1.78	0.05	1.53
1018	10	0.20	0.09	0.06	1.44	0.05	1.76	0.05	1.78	0.05	1.87	0.06	1.58
1199	11	0.15	0.15	0.10	1.50	0.08	1.96	0.07	2.08	0.07	2.13	0.08	1.90
1401	11	0.14	0.21	0.13	1.55	0.10	2.02	0.10	2.16	0.10	2.19	0.10	2.03
1601	11	0.15	0.30	0.20	1.53	0.16	1.94	0.15	2.07	0.15	2.07	0.15	1.97
1791	12	0.13	0.33	0.21	1.55	0.16	2.03	0.15	2.20	0.15	2.18	0.16	2.10
1996	9	0.13	0.38	0.25	1.54	0.19	2.03	0.17	2.21	0.18	2.18	0.18	2.12
2212	11	0.11	0.60	0.38	1.57	0.28	2.14	0.25	2.41	0.26	2.34	0.25	2.36
2406	8	0.10	0.72	0.46	1.58	0.33	2.17	0.29	2.46	0.30	2.38	0.29	2.45
2613	11	0.09	0.81	0.51	1.60	0.36	2.26	0.31	2.60	0.31	2.63	0.31	2.64
2802	9	0.08	0.92	0.57	1.60	0.40	2.31	0.34	2.72	0.34	2.72	0.33	2.77
3005	9	0.09	1.06	0.67	1.59	0.47	2.26	0.40	2.63	0.40	2.66	0.40	2.67
3202	9	0.09	1.30	0.83	1.58	0.59	2.22	0.51	2.55	0.50	2.59	0.50	2.60
3401	11	0.10	1.50	0.97	1.55	0.69	2.18	0.62	2.43	0.61	2.46	0.61	2.47
3603	10	0.09	1.52	0.96	1.58	0.66	2.30	0.57	2.65	0.56	2.69	0.56	2.72
3802	13	0.08	1.63	1.03	1.58	0.70	2.33	0.60	2.73	0.59	2.77	0.58	2.79
4002	8	0.10	1.91	1.19	1.61	0.87	2.19	0.77	2.48	0.77	2.47	0.76	2.50
4211	12	0.07	2.22	1.34	1.65	0.93	2.38	0.78	2.83	0.78	2.85	0.76	2.92
4401	12	0.07	2.60	1.55	1.68	1.05	2.47	0.87	3.00	0.87	3.00	0.84	3.11
4602	14	0.06	2.80	1.66	1.68	1.11	2.52	0.91	3.09	0.86	3.27	0.84	3.32
4806	12	0.07	3.38	2.03	1.66	1.38	2.45	1.14	2.95	1.11	3.05	1.10	3.08
5002	11	0.07	3.67	2.20	1.67	1.50	2.45	1.24	2.96	1.23	2.97	1.22	3.00
5199	12	0.06	3.84	2.28	1.68	1.51	2.54	1.20	3.19	1.14	3.35	1.16	3.30
5399	9	0.06	4.17	2.48	1.68	1.64	2.54	1.30	3.21	1.23	3.37	1.32	3.15
5598	9	0.05	4.37	2.59	1.69	1.69	2.58	1.33	3.29	1.25	3.49	1.27	3.45
5812	11	0.05	4.70	2.79	1.68	1.82	2.59	1.43	3.30	1.37	3.44	1.33	3.53
6005	12	0.06	5.03	3.02	1.67	2.00	2.52	1.59	3.17	1.58	3.18	1.51	3.32
6202	12	0.05	5.32	3.17	1.68	2.07	2.57	1.62	3.29	1.53	3.48	1.56	3.41
6396	12	0.05	5.70	3.41	1.67	2.21	2.58	1.72	3.32	1.65	3.47	1.67	3.41
6614	12	0.05	6.11	3.64	1.68	2.36	2.59	1.84	3.31	1.76	3.47	1.74	3.51
6802	11	0.05	6.44	3.85	1.67	2.50	2.58	1.96	3.29	1.84	3.51	1.84	3.50
7005	13	0.06	6.89	4.15	1.66	2.73	2.52	2.16	3.19	2.09	3.29	2.06	3.34
7205	14	0.07	7.81	4.82	1.62	3.29	2.37	2.74	2.85	2.67	2.93	2.62	2.98
7410	10	0.05	7.44	4.48	1.66	2.92	2.54	2.28	3.26	2.16	3.45	2.13	3.49
7602	10	0.05	7.76	4.67	1.66	3.05	2.55	2.39	3.25	2.24	3.46	2.48	3.13
7803	12	0.06	8.25	5.01	1.65	3.32	2.49	2.64	3.12	2.52	3.28	2.64	3.12
8003	8	0.07	9.29	5.82	1.60	4.00	2.32	3.37	2.75	3.27	2.84	3.75	2.48
8218	12	0.05	9.19	5.54	1.66	3.60	2.55	2.80	3.28	2.62	3.51	2.62	3.50
8411	12	0.05	10.69	6.41	1.67	4.12	2.60	3.16	3.38	2.93	3.64	2.92	3.66
8601	7	0.05	11.64	6.97	1.67	4.42	2.63	3.34	3.48	3.10	3.75	3.10	3.75
8802	14	0.05	13.16	7.91	1.66	5.08	2.59	3.88	3.39	3.64	3.61	3.82	3.45
9006	9	0.04	13.20	7.81	1.69	4.85	2.72	3.54	3.73	3.17	4.16	3.14	4.21
9202	12	0.04	13.78	8.20	1.68	5.08	2.71	3.68	3.74	3.30	4.17	3.47	3.97
9396	13	0.04	14.89	8.78	1.70	5.41	2.75	3.89	3.83	3.46	4.31	3.45	4.32
9603	12	0.05	17.08	10.25	1.67	6.44	2.65	5.12	3.33	4.77	3.58	5.05	3.38
9802	12	0.04	17.22	10.32	1.67	6.32	2.72	4.86	3.54	4.43	3.89	4.46	3.86
9998	13	0.04	17.09	10.06	1.70	5.96	2.87	4.41	3.88	3.90	4.39	3.90	4.38

Table A.22: Times for exponentiation in $\mathbb{F}_{2^n}$ for the test series *Sedimentary*.

Parallel exponentiation using a normal Gauß period of type (n, k)													
Parameter			seq.	parallel (nodes × processors)									
n	k	c		2	spd.	4	spd.	8	spd.	16	spd.	32	spd.
209	2	0.00	0.01	0.00	1.70	0.00	2.19	0.00	2.59	0.00	2.30	0.00	3.39
398	2	0.00	0.02	0.01	1.71	0.01	2.76	0.01	3.71	0.01	3.20	0.00	4.96
606	2	0.00	0.06	0.03	1.89	0.02	3.32	0.01	4.75	0.01	5.51	0.01	4.41
803	2	0.00	0.11	0.06	1.85	0.03	3.39	0.02	5.32	0.02	7.08	0.01	8.54
1018	1	0.00	0.07	0.04	1.81	0.02	3.29	0.01	5.13	0.01	6.33	0.01	6.26
1199	2	0.00	0.30	0.16	1.89	0.08	3.55	0.05	6.10	0.03	8.62	0.03	9.87
1401	2	0.00	0.42	0.22	1.89	0.12	3.59	0.07	6.42	0.04	9.93	0.04	10.57
1601	2	0.00	0.54	0.27	2.00	0.14	3.81	0.09	6.21	0.05	9.99	0.04	13.17
1791	2	0.00	0.67	0.33	2.05	0.17	3.95	0.10	6.94	0.06	10.61	0.06	11.88
1996	1	0.00	0.34	0.16	2.04	0.09	3.90	0.05	6.85	0.03	9.61	0.03	12.07
2212	1	0.00	0.49	0.25	2.00	0.13	3.84	0.07	6.93	0.05	10.42	0.04	13.68
2406	2	0.00	1.46	0.77	1.91	0.40	3.70	0.21	6.90	0.13	11.35	0.10	14.23
2613	2	0.00	1.80	0.93	1.93	0.48	3.77	0.26	6.96	0.15	11.59	0.12	15.55
2802	1	0.00	0.84	0.42	1.99	0.22	3.87	0.12	7.13	0.08	11.03	0.06	14.54
3005	2	0.00	2.31	1.22	1.89	0.63	3.69	0.33	6.90	0.20	11.50	0.15	15.52
3202	1	0.00	1.06	0.56	1.89	0.29	3.67	0.16	6.84	0.09	11.42	0.07	15.91
3401	2	0.00	3.03	1.61	1.88	0.83	3.65	0.44	6.92	0.25	12.02	0.17	18.09
3603	2	0.00	3.30	1.80	1.83	0.91	3.63	0.48	6.89	0.27	12.00	0.19	17.47
3802	1	0.00	1.39	0.77	1.82	0.39	3.57	0.21	6.73	0.12	11.40	0.09	15.37
4002	1	0.00	1.49	0.83	1.81	0.42	3.54	0.22	6.66	0.13	11.42	0.09	17.04
4211	2	0.00	4.92	2.47	1.99	1.27	3.87	0.72	6.80	0.41	12.09	0.27	18.53
4401	2	0.00	5.79	2.98	1.94	1.52	3.80	0.79	7.30	0.48	12.02	0.31	18.58
4602	1	0.00	2.45	1.25	1.96	0.65	3.78	0.34	7.19	0.20	12.22	0.14	17.10
4806	2	0.00	7.35	3.80	1.94	1.93	3.80	1.01	7.30	0.57	12.86	0.37	19.86
5002	1	0.00	3.04	1.56	1.95	0.80	3.80	0.42	7.24	0.25	12.35	0.16	18.66
5199	2	0.00	8.80	4.57	1.93	2.32	3.79	1.21	7.29	0.68	12.94	0.45	19.49
5399	2	0.00	9.61	5.02	1.91	2.54	3.78	1.33	7.23	0.74	13.06	0.50	19.08
5598	2	0.00	10.22	5.35	1.91	2.73	3.74	1.41	7.25	0.78	13.05	0.54	18.77
5812	1	0.00	4.13	2.17	1.91	1.10	3.74	0.58	7.14	0.33	12.69	0.22	18.50
6005	2	0.00	11.56	6.09	1.90	3.10	3.73	1.60	7.21	0.88	13.09	0.59	19.44
6202	1	0.00	4.68	2.46	1.90	1.25	3.73	0.66	7.14	0.37	12.78	0.26	18.14
6396	1	0.00	5.02	2.66	1.88	1.35	3.71	0.70	7.14	0.39	12.74	0.27	18.33
6614	2	0.00	14.38	7.60	1.89	3.87	3.71	1.99	7.24	1.08	13.27	0.72	19.87
6802	1	0.00	5.68	3.01	1.89	1.53	3.71	0.80	7.14	0.44	12.88	0.30	18.91
7005	2	0.00	15.93	8.46	1.88	4.28	3.72	2.20	7.23	1.22	13.10	0.82	19.43
7205	2	0.00	16.64	8.86	1.88	4.50	3.70	2.32	7.19	1.26	13.16	0.80	20.77
7410	1	0.00	6.53	3.49	1.87	1.77	3.68	0.92	7.11	0.51	12.81	0.33	19.57
7602	1	0.00	6.81	3.62	1.88	1.84	3.69	0.95	7.16	0.53	12.81	0.34	19.77
7803	2	0.00	19.00	10.08	1.88	5.11	3.72	2.63	7.23	1.44	13.16	0.93	20.40
8003	2	0.00	19.71	10.44	1.89	5.31	3.71	2.73	7.23	1.48	13.27	0.93	21.20
8218	1	0.00	8.20	4.31	1.90	2.19	3.75	1.13	7.24	0.62	13.21	0.40	20.26
8411	2	0.00	25.27	13.48	1.88	6.84	3.70	3.52	7.19	1.90	13.27	1.23	20.50
8601	2	0.00	27.90	14.90	1.87	7.56	3.69	3.88	7.20	2.11	13.20	1.35	20.64
8802	1	0.00	11.27	6.00	1.88	3.04	3.70	1.56	7.20	0.85	13.21	0.53	21.24
9006	2	0.00	33.25	17.50	1.90	8.85	3.76	4.55	7.31	2.45	13.56	1.53	21.74
9202	1	0.00	12.55	6.69	1.88	3.41	3.68	1.74	7.21	0.94	13.40	0.60	21.06
9396	1	0.00	13.52	7.20	1.88	3.67	3.69	1.88	7.18	1.01	13.36	0.65	20.81
9603	2	0.00	39.02	20.95	1.86	10.62	3.67	5.43	7.18	2.91	13.40	1.80	21.73
9802	1	0.00	14.97	7.98	1.88	4.04	3.70	2.08	7.19	1.12	13.36	0.70	21.52
9998	2	0.00	42.39	22.90	1.85	11.59	3.66	5.98	7.09	3.16	13.40	1.97	21.54

Table A.23: Times for exponentiation in $\mathbb{F}_{2^n}$ for the test series *PrimeGP*.

Comparison between expected and realized speedup												
Param.	*Arbitrary*			*Sparse*			*Sedimentary*			*PrimeGP*		
n	exp.	theo.	%	exp.	theo.	%	exp.	theo.	%	exp.	theo.	%
209	1.63	1.65	0.99	1.80	1.98	0.91	1.74	1.84	0.95	6.38	13.00	0.49
398	1.74	1.76	0.99	2.45	2.65	0.93	2.40	2.49	0.96	10.74	22.06	0.49
606	1.78	1.81	0.98	3.46	3.66	0.94	3.03	2.98	1.01	12.94	30.25	0.43
803	1.75	1.77	0.99	3.47	3.64	0.95	2.94	3.04	0.97	21.12	40.10	0.53
1018	1.70	1.72	0.99	3.38	3.52	0.96	3.26	3.36	0.97	16.29	50.85	0.32
1199	1.79	1.80	0.99	4.83	5.23	0.92	4.07	4.19	0.97	25.58	54.45	0.47
1401	1.77	1.79	0.99	5.13	5.58	0.92	4.28	4.51	0.95	28.41	63.64	0.45
1601	1.75	1.77	0.99	5.34	5.82	0.92	3.97	4.15	0.96	36.46	72.73	0.50
1791	1.73	1.75	0.99	5.32	5.72	0.93	4.45	4.65	0.96	33.61	81.36	0.41
1996	1.69	1.71	0.99	5.22	5.65	0.92	4.55	4.75	0.96	34.59	90.68	0.38
2212	1.75	1.77	0.99	6.62	7.23	0.92	5.17	5.38	0.96	40.34	92.12	0.44
2406	1.76	1.78	0.99	6.31	6.78	0.93	5.43	5.70	0.95	43.50	100.21	0.43
2613	1.75	1.77	0.99	6.59	7.02	0.94	6.02	6.34	0.95	46.80	108.83	0.43
2802	1.74	1.76	0.99	6.84	7.23	0.95	6.45	6.83	0.94	44.26	116.71	0.38
3005	1.73	1.74	0.99	6.79	7.14	0.95	6.21	6.48	0.96	48.37	125.17	0.39
3202	1.74	1.75	0.99	7.10	7.45	0.95	6.03	6.26	0.96	49.28	133.38	0.37
3401	1.72	1.73	0.99	8.18	8.57	0.95	5.66	5.82	0.97	57.48	141.67	0.41
3603	1.71	1.72	1.00	7.03	7.39	0.95	6.45	6.70	0.96	55.55	150.08	0.37
3802	1.69	1.70	0.99	7.03	7.43	0.95	6.73	7.13	0.94	48.85	158.38	0.31
4002	1.68	1.69	0.99	6.90	7.23	0.95	5.83	6.07	0.96	54.57	166.71	0.33
4211	1.72	1.74	0.99	7.94	8.43	0.94	7.22	7.57	0.95	60.51	161.92	0.37
4401	1.72	1.74	0.99	10.01	10.74	0.93	7.88	8.34	0.94	61.08	169.23	0.36
4602	1.73	1.74	0.99	10.51	10.99	0.96	8.67	8.93	0.97	57.56	176.96	0.33
4806	1.74	1.75	0.99	11.05	11.57	0.95	7.98	8.21	0.97	66.88	184.81	0.36
5002	1.73	1.75	0.99	9.57	10.03	0.95	7.78	8.25	0.94	63.12	192.35	0.33
5199	1.74	1.76	0.99	11.54	12.07	0.96	8.94	9.30	0.96	66.10	199.92	0.33
5399	1.74	1.75	0.99	11.67	12.32	0.95	9.08	9.52	0.95	64.87	207.62	0.31
5598	1.74	1.76	0.99	11.65	12.32	0.95	9.54	10.04	0.95	64.60	215.27	0.30
5812	1.74	1.75	1.00	11.62	12.45	0.93	9.68	10.11	0.96	64.11	223.50	0.29
6005	1.73	1.74	0.99	10.04	10.51	0.95	9.05	9.46	0.96	67.63	230.92	0.29
6202	1.74	1.76	0.99	11.83	12.47	0.95	9.66	10.18	0.95	64.30	238.50	0.27
6396	1.72	1.74	0.99	12.02	12.73	0.94	9.68	10.37	0.93	65.12	245.96	0.26
6614	1.73	1.75	0.99	12.05	12.77	0.94	9.84	10.39	0.95	69.43	254.35	0.27
6802	1.77	1.78	0.99	10.35	11.00	0.94	9.89	10.41	0.95	67.06	261.58	0.26
7005	1.73	1.75	0.99	10.20	10.82	0.94	9.31	9.83	0.95	68.24	269.38	0.25
7205	1.73	1.75	0.99	10.22	11.07	0.92	8.03	8.14	0.99	73.24	277.08	0.26
7410	1.73	1.74	0.99	11.67	12.75	0.92	9.93	10.28	0.97	70.47	284.96	0.25
7602	1.73	1.75	0.99	11.27	12.54	0.90	9.77	10.16	0.96	71.18	292.35	0.24
7803	1.73	1.75	0.99	10.13	10.58	0.96	9.19	9.55	0.96	72.37	300.08	0.24
8003	1.73	1.75	0.99	9.65	10.37	0.93	7.61	7.89	0.96	75.61	307.77	0.25
8218	1.74	1.75	0.99	12.01	12.83	0.94	10.02	10.51	0.95	72.37	293.46	0.25
8411	1.75	1.76	0.99	11.52	12.47	0.92	10.56	10.97	0.96	73.45	300.36	0.24
8601	1.74	1.76	0.99	14.23	15.20	0.94	11.00	11.66	0.94	74.26	307.14	0.24
8802	1.74	1.75	0.99	15.00	15.99	0.94	10.59	11.09	0.95	76.87	314.32	0.24
9006	1.74	1.75	0.99	15.64	16.35	0.96	12.70	13.31	0.95	76.83	321.61	0.24
9202	1.73	1.75	0.99	15.73	16.46	0.96	12.61	13.38	0.94	75.81	328.61	0.23
9396	1.75	1.76	0.99	15.92	17.13	0.93	13.15	13.83	0.95	75.31	335.54	0.22
9603	1.75	1.76	0.99	13.76	14.69	0.94	10.38	10.93	0.95	78.87	342.93	0.23
9802	1.74	1.75	0.99	14.21	15.09	0.94	11.61	12.09	0.96	78.13	350.04	0.22
9998	1.75	1.76	0.99	15.97	17.40	0.92	13.43	14.14	0.95	78.99	357.04	0.22

Table A.24: Comparison of expected and realized speed-up.

References

G. B. Agnew, R. C. Mullin, I. M. Onyszchuk & S. A. Vanstone (1991). An implementation for a fast public–key cryptosystem. *Journal of Cryptology* **3**, 63–79.

G. B. Agnew, R. C. Mullin & S. A. Vanstone (1988). Fast Exponentiation in $GF(2^n)$. In *Advances in Cryptology: Proceedings of EUROCRYPT 1988*, C. G. Günther, editor, number 330 in Lecture Notes in Computer Science, 251–255. Springer-Verlag, Berlin.

G. B. Agnew, R. C. Mullin & S. A. Vanstone (1993). An Implementation of Elliptic Curve Cryptosystems Over F_{2155}. *IEEE Journal on Selected Areas in Communications* **11**(5), 804–813.

Alfred V. Aho, John E. Hopcroft & Jeffrey D. Ullman (1974). *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading MA.

Y. Asano, T. Itoh & S. Tsujii (1989). Generalised fast algorithm for computing multiplicative inverses in $GF(2^m)$. *Electronics Letters* **25**(10), 664–665.

D.W. Ash, I.F. Blake & S.A. Vanstone (1989). Low complexity normal bases. *Discrete Applied Mathematics* **25**, 191–210.

Albert H. Beiler (1966). *Recreations in the Theory of Numbers: The Queen of Mathematics Entertains*. Dover Publications, Inc., New York.

M. Ben-Or (1981). Probabilistic algorithms in finite fields. In *Proceedings of the 22nd Annual IEEE Symposium on Foundations of Computer Science*, Nashville TN, 394–398.

T. Beth, W. Geiselmann & F. Meyer (1991). Finding (Good) Normal Bases in Finite Fields. In *Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation ISSAC '91*, Bonn, Germany, Stephen M. Watt, editor, 173–178. ACM Press.

I. F. Blake, R. Fuji-Haro, R. C. Mullin & S. A. Vanstone (1984). Computing logarithms in finite fields of characteristic two. *SIAM Journal on Algebraic and Discrete Methods* **5**(2), 276–285.

Ian F. Blake, Ron M. Roth & Gadiel Seroussi (1998). Efficient Arithmetic in $GF(2^n)$ through Palindromic Representation. Technical Report HPL-98-134, Visual Computing Department, Hewlett Packard Laboratories. Available via `www.hpl.hp.com/techreports/98/HPL-98-134.html`.

A. Borodin & I. Munro (1975). *The Computational Complexity of Algebraic and Numeric Problems*. Number 1 in Theory of computation series. American Elsevier Publishing Company, New York.

J. Bos & M. Coster (1990). Addition Chain Heuristics. In *Advances in Cryptology: Proceedings of CRYPTO '89*, Santa Barbara CA, 400–407.

Siegfried Bosch (1993). *Algebra.* Springer Verlag, Berlin.

A. Brauer (1939). On addition chains. *Bulletin of the American Mathematical Society* **45**, 736–739.

Ernest F. Brickell, D. Gordon, K. McCurley & D. Wilson (1993). Fast Exponentiation with Precomputation. In *Advances in Cryptology: Proceedings of EUROCRYPT 1992,* Balatonfüred, Hungary, R. Rueppel, editor, number 658 in Lecture Notes in Computer Science, 200–207. Springer-Verlag, Berlin.

David G. Cantor (1989). On Arithmetical Algorithms over Finite Fields. *Journal of Combinatorial Theory, Series A* **50**, 285–300.

Che Wun Chiou (1993). Parallel implementation of the RSA public-key cryptosystem. *International Journal of Computer Mathematics* **48**, 153–155.

D. Coppersmith (1984). Fast Evaluation of Logarithms in Fields of Characteristic Two. *IEEE Transactions on Information Theory* **IT-30**(4), 587–594.

D. Coppersmith & J. H. Davenport (1985). An application of factoring. *Journal of Symbolic Computation* **1**, 241–243.

Robert M. Corless, Gaston H. Gonnet, D. E. G. Hare & David J. Jeffrey (1993). Lambert's W Function in Maple. *Maple Technical Newsletter* .

Erik De Win, Antoon Bosselaers, Servaas Vandenberghe, Peter De Gersem & Joos Vandewalle (1996). A Fast Software Implementation for Arithmetic Operations in $GF(2^n)$. In *Advances in Cryptology: Proceedings of ASIACRYPT 1996,* Kyongju, Korea, Kwangjo Kim, editor, number 1163 in Lecture Notes in Computer Science, 65–76. Springer-Verlag. Also available via `ftp://ftp.esat.kuleuven.ac.be/pub/cosic/dewin/asia96p103.ps.gz`.

Whitfield Diffie & Martin E. Hellman (1976). New directions in cryptography. *IEEE Transactions on Information Theory* **IT-22**(6), 644–654.

T. ElGamal (1985). A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory* **IT-31**(4), 469–472.

P. Erdös (1960). Remarks on Number theory III: On addition chains. *Acta Arithmetica* **6**, 77–81.

Sandra Feisel, Joachim von zur Gathen & M. Amin Shokrollahi (1999). Normal bases via general Gauß periods. *Mathematics of Computation* **68**(225), 271–290.

S. Gao & H. W. Lenstra, Jr. (1992). Optimal normal bases. *Designs, Codes, and Cryptography* **2**, 315–323.

Shuhong Gao (2001). Abelian Groups, Gauss Periods, and Normal Bases. *Finite Fields and Their Applications* **7**(1), 148–164.

Shuhong Gao, Joachim von zur Gathen & Daniel Panario (1995). Gauss periods and fast exponentiation in finite fields. In *Proceedings of LATIN '95,* Valparaíso, Chile, number 911 in Lecture Notes in Computer Science, 311–322. Springer-Verlag.

Shuhong Gao, Joachim von zur Gathen, Daniel Panario & Victor Shoup (2000). Algorithms for exponentiation in finite fields. *Journal of Symbolic Computation* **29**(6), 879–889.

Shuhong Gao, Jason Howell & Daniel Panario (1999). Irreducible polynomials of given forms. *Contemporary Mathematics* **225**, 43–54.

Shuhong Gao & Daniel Panario (1997). Tests and Constructions of Irreducible Polynomials over Finite Fields. In *Foundations of Computational Mathematics*, Felipe Cucker & Michael Shub, editors, 346–361. Springer Verlag.

Joachim von zur Gathen (1991). Efficient and optimal exponentiation in finite fields. *computational complexity* **1**, 360–394.

Joachim von zur Gathen (1992). Processor-efficient exponentiation in finite fields. *Information Processing Letters* **41**, 81–86.

Joachim von zur Gathen & Jürgen Gerhard (1996). Arithmetic and Factorization of Polynomials over $\mathbb{F}_2$. In *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation ISSAC '96,* Zürich, Switzerland, Y. N. Lakshman, editor, 1–9. ACM Press. URL `http://www-math.uni-paderborn.de/~aggathen/Publications/polyfactTR.ps`. Technical report tr-rsfb-96-018, University of Paderborn, Germany, 1996, 43 pages.

Joachim von zur Gathen & Jürgen Gerhard (1999). *Modern Computer Algebra.* Cambridge University Press, Cambridge, UK. ISBN 0-521-64176-4. URL `http://www-math.uni-paderborn.de/~aggathen/mca/`.

Joachim von zur Gathen & Michael Nöcker (1997). Exponentiation in Finite Fields: Theory and Practice. In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes: AAECC-12,* Toulouse, France, Teo Mora & Harold Mattson, editors, number 1255 in Lecture Notes in Computer Science, 88–113. Springer-Verlag.

Joachim von zur Gathen & Michael Nöcker (1999). Computing Special Powers in Finite Fields: Extended Abstract. In *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation ISSAC '99,* Vancouver, Canada, Sam Dooley, editor, 83–90. ACM Press.

Joachim von zur Gathen & Michael Nöcker (2000). Exponentiation using addition chains for finite fields. Submitted.

Joachim von zur Gathen & Sandra Schlink (1996). Normal bases via general Gauss periods. Reihe Informatik tr-ri-96-177, Universität-Gesamthochschule Paderborn.

Carl Friedrich Gauss (1801). *Disquisitiones Arithmeticae*. Gerh. Fleischer Iun., Leipzig. English translation by Arthur A. Clarke, Springer-Verlag, New York, 1986.

W. Geiselmann (1994). *Algebraische Algorithmenentwicklung am Beispiel der Arithmetik in endlichen Körpern*. Dissertation, Universität Karlsruhe, Aachen.

Alan Gibbons & Wojciech Rytter (1988). *Efficient parallel algorithms*. Cambridge University Press.

Solomon W. Golomb (1967). *Shift Register Sequences*. Holden-Day Series in Information Systems. Holden-Day, Inc., San Francisco, California. With portions coauthored by Lloyd R. Welch, Richard M. Goldstein, and Alfred W. Hales.

Daniel M. Gordon (1998). A Survey of Fast Exponentiation Methods. *Journal of Algorithms* **27**, 129–146.

Daniel M. Gordon & Kevin S. McCurley (1992). Massively Parallel Computation of Discrete Logarithms. In *Advances in Cryptology: Proceedings of CRYPTO '92*, Santa Barbara CA, Ernest F. Brickell, editor, number 740 in Lecture Notes in Computer Science, 312–323. Springer-Verlag.

Dirk Hachenberger (1997). *Finite Fields: Normal Bases and Completely Free Elements*. The Kluwer international series in engineering and computer science. Kluwer Academic Publishers, Boston/Dordrecht/London.

Walter Hansen (1959). Zum Scholz-Brauerschen Problem. *Journal für die Reine und Angewandte Mathematik* **202**(3/4), 129–136.

H. Hasse (1980). *Number Theory*. Number 229 in Grundlehren der mathematischen Wissenschaften. Springer-Verlag.

Helmut Hasse (1969). *Zahlentheorie*. Akademie-Verlag, Berlin, 3rd edition. English translation: Hasse (1980).

Kurt Hensel (1888). Über die Darstellung der Zahlen eines Gattungsbereiches für einen beliebigen Primdivisor. *Journal für die Reine und Angewandte Mathematik* **103**, 230–237.

IEEE (2000). IEEE Standard Specifications for Public-Key Cryptography. Technical Report IEEE Std 1363-2000, Institute of Electrical and Electronics Engineers, Inc., 3 Park Avenue, New York, NY 10016-5997, USA.

T. Itoh & S. Tsujii (1988a). Effictive recursive algorithm for computing multiplicative inverses in $GF(2^m)$. *Electronics Letters* **24**(6), 334–335.

T. Itoh & S. Tsujii (1988b). A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Bases. *Information and Computation* **78**, 171–177.

Nathan Jacobson (1974). *Basic Algebra I*. Freeman, San Francisco.

D. JUNGNICKEL (1993). *Finite Fields: Structure and Arithmetics*. BI Wissenschaftsverlag, Mannheim.

A. KARATSUBA & YU. OFMAN (1963). Multiplication of multidigit numbers on automata. *Soviet Physics–Doklady* **7**(7), 595–596. Translated from Doklady Akademii Nauk SSSR, Vol. 145, No. 2, pp. 293–294, July, 1962.

DONALD E. KNUTH (1962). Evaluation of Polynomials By Computer. *Communications of the ACM* **5**(1), 595–599.

DONALD E. KNUTH (1998). *The Art of Computer Programming, vol. 2, Seminumerical Algorithms*. Addison-Wesley, Reading MA, 3rd edition. First edition 1969.

H. T. KUNG (1976). New Algorithms and Lower Bounds for the Parallel Evaluation of Certain Rational Expressions and Recurrences. *Journal of the ACM* **23**(2), 252–261.

RUDOLF LIDL & HARALD NIEDERREITER (1983). *Finite Fields*. Number 20 in Encyclopedia of Mathematics and its Applications. Addison-Wesley, Reading MA.

UELI M. MAURER & STEFAN WOLF (1999). The relationship between breaking the Diffie-Hellman protocol and computing discrete logarithms. *SIAM Journal on Computing* **28**(5), 1689–1721.

KEVIN S. MCCURLEY (1990). The Discrete Logarithm Problem. In *Cryptology and Computational Number Theory*, CARL POMERANCE, editor, number 42 in Proceedings of Symposia in Applied Mathematics, 49–74. American Mathematical Society.

ALFRED J. MENEZES, IAN F. BLAKE, XUHONG GAO, RONALD C. MULLIN, SCOTT A. VANSTONE & TOMIK YAGHOOBIAN (1993). *Applications of finite fields*. Kluwer Academic Publishers, Norwell MA.

R. T. MOENCK (1973). Fast computation of gcd's. In *Proceedings of the Fifth Annual ACM Symposium on the Theory of Computing,* Austin TX, 142–151. ACM Press.

R. C. MULLIN, I. M. ONYSZCHUK, S. A. VANSTONE & R. M. WILSON (1989). Optimal normal bases in GF(p^n). *Discrete Applied Mathematics* **22**, 149–161.

MICHAEL NÖCKER (2000). Some Remarks on Parallel Exponentiation: Extended Abstract. In *Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation ISSAC2000*, St. Andrews, Scotland, CARLO TRAVERSO, editor, 250–257.

A. ODLYZKO (1985). Discrete Logarithms and their cryptographic significance. In *Advances in Cryptology: Proceedings of EUROCRYPT 1984,* Paris, France, 224–314. Springer-Verlag.

A. M. ODLYZKO (1994). Discrete Logarithms and Smooth Polynomials. In *Finite Fields: Theory, Applications, and Algorithms*, GARY L. MULLEN & PETER JAU-SHYONG SHINE, editors, volume 168 of *Contemporary Mathematics*, 269–278.

Andrew Odlyzko (2000). Discrete Logarithms: The Past and the Future. *Designs, Codes, and Cryptography* **19**, 129–145. URL `http://www.research.att.com/~amo/doc/discrete.logs.future.ps`.

Jimmy K. Omura & James L. Massey (1986). Computational method and apparatus for finite field arithmetic. *United States Patent 4,587,627* Date of Patent: May 6, 1986, available via `http://www.patents.ibm.com/details?pn=US04587627_`.

Ivan M. Onyszchuk, Ronald C. Mullin & Scott A. Vanstone (1988). Computational method and apparatus for finite field multiplication. *United States Patent* **4,745,568**. Date of Patent: May 17, 1988, available via `www.patents.ibm.com/details?pn=US04745568_`.

Paul C. van Oorshot (1992). A Comparison of Practical Public Key Cryptosystems Based on Integer Factorization and Discrete Logarithms. In *Contemporary Cryptology, The Science of Information Integrity*, Gustavus J. Simmons, editor, chapter 5, 289–322. IEEE Press, New York.

Stephen C. Pohlig & Martin E. Hellman (1978). An Improved Algorithm for Computing Logarithms over $GF(p)$ and Its Cryptographic Significance. *IEEE Transactions on Information Theory* **IT-24**(1), 106–110.

R. Ree (1971). Proof of a Conjecture of S. Chowla. *Journal of Number Theory* **3**, 210–212.

R. L. Rivest, A. Shamir & L. M. Adleman (1978). A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM* **21**(2), 120–126.

Tony Rosati (1989). A high speed data encryption processor for public–key cryptography. In *Proc. of IEEE Custom Integrated Circuits Conference,* San Diego CA, 12.3.1 – 12.3.5.

Michael Rosing (1999). *Implementing elliptic curve cryptography.* Manning Publications Co., Greenwich, CT.

A. Scholz (1937). Aufgabe 253. *Jahresberichte der DMV* **47**, 41–42.

A. Schönhage (1971). Schnelle Berechnung von Kettenbruchentwicklungen. *Acta Informatica* **1**, 139–144.

A. Schönhage (1975). A lower bound for the length of addition chains. *Theoretical Computer Science* **1**, 1–12.

A. Schönhage (1977). Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2. *Acta Informatica* **7**, 395–398.

A. Schönhage & V. Strassen (1971). Schnelle Multiplikation großer Zahlen. *Computing* **7**, 281–292.

Richard Schroeppel, Hilarie Orman & Sean O'Malley (1995a). Fast Key Exchange with Elliptic Curve Systems. Technical report, Department of Computer Science, The University of Arizona, Tucson, AZ 85721. URL `http://www.cs.arizona.edu/~rcs/ecrv/`. A version of this has been published at CRYPTO'95, see Schroeppel *et al.* (1995b).

Richard Schroeppel, Hilarie Orman, Sean O'Malley & Oliver Spatscheck (1995b). Fast Key Exchange with Elliptic Curve Systems. In *Advances in Cryptology: Proceedings of CRYPTO '95*, Santa Barbara CA, Don Coppersmith, editor, number 963 in Lecture Notes in Computer Science, 43–56. Springer.

I. A. Semaev (1989). Construction of Polynomials Irreducible Over a Finite Field with Linearly Independent Roots. *Mathematics of the USSR Sbornik* **63**(2), 507–519.

D. R. Stinson (1990). Some Observations on Parallel Algorithms for Fast Exponentiation in $GF(2^n)$. *SIAM Journal on Computing* **19**(4), 711–717.

Douglas R. Stinson (1995). *Cryptography, Theory and Practice.* CRC Press Inc., Boca Raton FL.

Richard G. Swan (1962). Factorization of polynomials over finite fields. *Pacific Journal of Mathematics* **12**, 1099–1106.

L. G. Valiant (1990). A bridging model for parallel computation. *Communications of the ACM* **33**, 103–111.

C. C. Wang, T. K. Truong, H. M. Shao, L. J. Deutsch, J. K. Omura & I. S. Reed (1985). VLSI Architectures for Computing Multiplications and Inverses in $GF(2^m)$. *IEEE Transactions on Computers* **C-34**, 709–717.

Alfred Wassermann (1990). Konstruktion von Normalbasen. *Bayreuther Math. Schriften* **31**, 155–164.

Alfred Wassermann (1993). Zur Arithmetik in endlichen Körpern. *Bayreuther Math. Schriften* **44**, 147–251.

Dazhuan Xu (1990). A fast algorithm for multiplicative inverses based on the normal basis representation. *Journal of Nanjing Aeronautical Institute (English edition)* **7**(1), 121–124.

Andrew Chi-Chih Yao (1976). On the Evaluation of Powers. *SIAM Journal on Computing* **5**(1), 100–103.

Neal Zierler & John Brillhart (1968). On Primitive Trinomials (Mod 2). *Information and Control* **13**, 541–554.

Neal Zierler & John Brillhart (1969). On Primitive Trinomials (Mod 2), II. *Information and Control* **14**, 566–569.