PADERBORN UNIVERSITY
The University for the Information Society

Faculty for Computer Science, Electrical Engineering and Mathematics

Computing on Encrypted Data using Trusted Execution Environments

Andreas Fischer

Dissertation

submitted in partial fulfillment
of the requirements for the degree of

Doktor der Naturwissenschaften (Dr. rer. nat.)

Advisors

Prof. Dr. Eric Bodden
Prof. Dr. Florian Kerschbaum

Paderborn, April 27, 2021

Abstract

Cloud services provide on-demand access to cost-efficient computer resources such as data storage and computing power. However, when using these services, data is at risk of being stolen by attackers observing the cloud, e.g., malicious administrators or external intruders. To ensure the confidentiality of sensitive data, encryption can be applied prior to transferring it to the cloud. However, in order to use the cloud's computing power without compromising data confidentiality, the cloud must compute on encrypted data.

Secure and efficient computation on encrypted data is difficult. Common symmetric encryption schemes can be used to achieve confidentiality for outsourced data but prevent untrusted cloud service providers from computing on ciphertexts. Fully homomorphic encryption (FHE) can perform arbitrary computations on ciphertexts, but suffers high computational costs. Hardware-based trusted execution environments such as Intel Software Guard Extensions (SGX) entail only little computational overhead but are vulnerable to side-channel attacks which can completely dismantle their security guarantees.

This dissertation presents a novel architecture combining cryptographic primitives with hardware-based security to compute on encrypted data. Compared to a solution solely relying on SGX, our approach provides a program-independent trusted code base implemented in a small trusted module, which can be reused across programs and hardened against side-channels. Thus, our approach significantly reduces the surface for attacks exploiting software vulnerabilities in the protected application. Compared to only using FHE, our approach provides high efficiency due to fast ciphertext operations and support for control flow.

We consider active attackers controlling the cloud server capable of altering the control flow of the outsourced program and maliciously interacting with the trusted module. We show that, if the attacker is allowed to arbitrarily interact with the trusted module, it can amplify leakage using adaptive queries. As a defense for this dataflow modification attack, we introduce dataflow authentication (DFAuth), which we implement using the complementary concepts of homomorphic authenticated symmetric encryption (HASE) and trusted authenticated ciphertext operations (TACO).

We also address the concern that information revealed through control flow might be inacceptable from a security perspective. To avoid sacrificing performance for security, we analyze the trade-off between the two in more detail. We use quantitative information flow to measure leakage, running time to measure performance and program transformation techniques to alter the trade-off between the two. Combined with information flow policies, we formalize the problem of policy-aware security and performance trade-off (PASAPTO) analysis, prove its NP-hardness and present two algorithms solving it heuristically.

Zusammenfassung

Cloud-Dienste bieten Zugang zu kostengünstiger IT-Infrastruktur wie Speicherplatz und Rechenleistung. Bei der Nutzung dieser Dienste besteht jedoch die Gefahr, dass Daten von Angreifern gestohlen werden. Um die Vertraulichkeit sensibler Daten zu gewährleisten, können sie vor der Übertragung verschlüsselt werden. Um jedoch die Rechenleistung der Cloud zu nutzen, ohne die Vertraulichkeit zu beeinträchtigen, muss die Cloud mit verschlüsselten Daten rechnen.

Sicheres und effizientes Rechnen mit verschlüsselten Daten ist eine Herausforderung. Herkömmliche Verschlüsselungsverfahren verhindern, dass ein nicht vertrauenswürdiger Dienstanbieter mit Chiffraten rechnet. Vollhomomorphe Verfahren erlauben beliebige Berechnungen mit Chiffraten, verursachen jedoch hohe Rechenkosten. Auf Hardware-basierter Sicherheit beruhende Ansätze wie Intel SGX verursachen nur geringen Rechenaufwand, sind jedoch anfällig für Seitenkanalangriffe, die die Sicherheitsmechanismen außer Kraft setzen können.

Diese Dissertation präsentiert eine Architektur, die kryptografische Verfahren mit einem Hardwaresicherheitsmodul kombiniert, um mit verschlüsselten Daten zu rechnen. Im Vergleich zu einer rein auf Hardware-basierter Sicherheit beruhenden Lösung bietet der vorgestellte Ansatz eine minimale und programmunabhängige Softwarebasis, die über alle Anwendungen hinweg wiederverwendet werden kann. Somit kann die Angriffsfläche für Softwareschwachstellen in der geschützten Anwendung erheblich reduziert werden. Im Vergleich zu einer ausschließlich auf vollhomomorpher Verschlüsselung basierenden Lösung wird eine hohe Effizienz erreicht, da Kontrollflussentscheidungen unterstützt werden und Chiffrat-Operationen erheblich effizienter sind.

Einerseits werden aktive Angreifer betrachtet, die den Cloud-Server kontrollieren und den Kontrollfluss des ausgelagerten Programms manipulieren sowie böswillig mit dem Sicherheitsmodul interagieren können. Kann ein Angreifer beliebig mit dem Sicherheitsmodul interagieren, so kann er die Verschlüsselung durch adaptive Anfragen an das Sicherheitsmodul brechen. Um diesen Angriff der Datenflussmodifikation zu verhindern, wird in dieser Arbeit das Konzept der Datenflussauthentifizierung vorgestellt.

Andererseits wird betrachtet, dass die Offenlegung von Kontrollfluss aus Sicherheitsgründen unerwünscht sein kann. Um nicht leichtfertig Leistung für Sicherheit zu opfern, wird der Kompromiss zwischen den beiden Eigenschaften genauer analysiert. Die vorgestelle Analyse quantifiziert sowohl die Laufzeit eines Programms als auch den aus Kontrollfluss resultierenden Informationsfluss und verändert den Kompromiss über Programmtransformationstechniken. In Kombination mit Informationsflussregeln formuliert diese Arbeit die resultierende Analyse als Optimierungsproblem, zeigt die NP-schwere des zugehörigen Entscheidungsproblems und präsentiert zwei Heuristiken zur effizienten Lösung.

To my family

Acknowledgments

This dissertation is the result of my work as a PhD student at SAP Security Research and Paderborn University. I would like to take this opportunity to thank everyone who has contributed to the completion of this work.

First and foremost, I express my sincere gratitude to Prof. Dr. Florian Kerschbaum for believing in my abilities and allowing me to pursue this adventure. I learned a lot from him and the quickness of his mind surprises me to this day. Without his continued guidance this endeavor would surely not have been a success.

Moreover, I especially thank Prof. Dr. Eric Bodden for accepting me as a PhD student at Paderborn University and providing valuable feedback to my work. Many thanks also for allowing me to spend a few weeks at the Heinz Nixdorf Institute and introducing me to the rest of the Secure Software Engineering group.

I also thank Prof. Dr. Johannes Blömer, Prof. Dr. Ben Hermann and Prof. Dr. Juraj Somorovsky for joining the examination committee. A big thank you also to Prof. Dr. Jörn Müller-Quade for introducing me to IT security and cryptography in the first place. A special thanks to Dr. Marion Baumann for constantly encouraging me to finish this project.

I am grateful to all SAP colleagues who supported me over the years. I would like to especially mention my co-authors Dr. Benny Fuhry, Jonas Janneck, Jörn Kußmaul, and Nikolas Krätzschmar. Thank you for your continued support and your valuable contributions to my research. For many fruitful discussions I also thank my colleagues from the applied cryptography group Dr. Florian Hahn and Dr. Anselme Kemgne Tueno. I also thank my other fellow PhD students Daniel Bernau, Jonas Böhler, and Benjamin Weggenmann for many interesting discussions and coffee breaks.

Last but not least, I express my deepest gratitude to my family. In particular, I thank my parents, Karin and Uwe Fischer, for their love and support in anything I do. I thank my brother, Alexander Fischer, for many enjoyable evenings and providing the necessary distraction. My late great-uncle Stefan Rempold also deserves a special thanks for allowing me to play around with his electronics and computers when I was a child. Finally, I thank my girlfriend Maike, who has been supporting me in everything I do for more than ten years. Without her encouragement, love and patience this dissertation would never have been finished.

Contents

List of Figures

List of Tables

List of Definitions

List of Theorems

List of Constructions

List of Algorithms

Abbreviations

0-1-ILP 0-1 integer linear programming

AE authenticated encryption

AES Advanced Encryption Standard

CPU central processing unit

CRT Chinese remainder theorem

CSP cloud service provider

DDH decisional Diffie-Hellman

DFATA DFAuth trade-off analyzer

DFAuth dataflow authentication

DNF disjunctive normal form

EV electric vehicle

FE functional encryption

FHE fully homomorphic encryption

GCM Galois/counter mode

HASE homomorphic authenticated symmetric encryption

HE homomorphic encryption

HVI hypervolume indicator

IND-CCA indistinguishable encryptions under a chosen ciphertext attack

IND-CPA indistinguishable encryptions under a chosen plaintext attack

iO indistinguishability obfuscation

IT information technology

MAC message authentication code

MOP	multi-objective optimization problem
MPC	multi-party computation
OPE	order-preserving encryption
ORAM	oblivious RAM
ORE	order-revealing encryption
PASAPTO	policy-aware security and performance trade-off
PDG	program dependency graph
PEKS	public-key encryption with keyword search
PHE	partially homomorphic encryption
PIR	private information retrieval
PPE	property-preserving encryption
PPT	probabilistic polynomial time
PRF	pseudorandom function
QIF	quantitative information flow
RAM	random access memory
ROP	return-oriented programming
SDK	software development kit
SGX	Software Guard Extensions
SIV	synthetic initialization vector
SSA	single static assignment
SSE	searchable symmetric encryption
SWHE	somewhat homomorphic encryption
TACO	trusted authenticated ciphertext operations
TCB	trusted code base
TEE	trusted execution environment
UF-CPA	unforgeability under a chosen plaintext attack

1 Introduction

This dissertation presents a novel architecture combining cryptographic primitives with hardware-based security to compute on encrypted data. Section 1.1 discusses the motivation of our work. Section 1.2 summarizes our scientific contributions and Section 1.3 describes the structure of the remainder of this work.

1.1 Motivation

At the end of the 20th century the basis of our global economy started to rapidly shift from traditional industries such as oil and steel towards information technology (IT). In 2020, data is regarded as “the new oil” and companies either are in the IT business or heavily rely on IT for everyday operation. At the same time, businesses are increasingly trying to reduce operational costs for IT infrastructure by outsourcing to third parties.

This outsourcing trend is in particular fueled by the widespread availability of modern cloud infrastructure. Essentially, cloud computing allows computer resources such as data storage and computing power to be consumed over computer networks similar to other standardized commodities like electricity, water and gas. Benefits for cloud customers include consumption-based billing and rapid elasticity, that is, the resource allocation can be continuously adjusted to work load demand. Cloud service providers (CSPs) on the other hand can leverage economies of scale in data center operation and hardware acquisition.

Similar effects of scale apply to IT security mechanisms utilized by CSPs. For example, the knowledge of security experts can be used to improve the cloud infrastructure at large and to defend against attacks for all cloud customers. Frequently deployed tools include security information and event management systems, virus scanners, network firewalls and program analysis. However, none of these security mechanisms protect against attackers operating from inside the CSP. For example, a malicious employee may attempt to obtain trade secrets of cloud customers and sell them to competitors. In case provider and customer do not share the same jurisdiction, the customer may also fear involuntary sharing of data with foreign governments caused by legal requirements in the jurisdiction of the provider.

To ensure the confidentiality of sensitive data, customers can encrypt their data prior to transferring it to the cloud. For encryption to protect against insider attacks, as a minimum requirement it must be ensured that the CSP does not have access to the decryption key at any time. Consequently, if a customer wants to use the cloud’s computing power without compromising data confidentiality, the cloud must compute on encrypted data. However, secure and efficient computation on encrypted data is difficult.

The structure of common encryption schemes such as the Advanced Encryption Standard (AES) [Aes] inherently prevents computations on ciphertexts. Thus, these schemes can only be used in cloud storage use cases in which the CSP stores data but does not perform any other operations on it. The CSP then only has to be trusted with regards to availability, whereas confidentiality is achieved using encryption. Secure file stores can further improve availability by incorporating redundancy techniques allowing data to be distributed over multiple CSPs. Still, no operations other than storage and retrieval are supported on encrypted data.

Homomorphic encryption (HE) is a class of encryption that supports computations on ciphertexts. Constructions allowing limited operation on ciphertexts, e.g., only integer multiplication, have been known since the late 1970s [RSA78]. The concept of fully homomorphic encryption (FHE), i.e., encryption schemes supporting arbitrary computations on ciphertexts, essentially has been proposed approximately at the same time [RAD78], but the first construction was not discovered before 2009 [Gen09]. Although FHE is sometimes called "the holy grail of cryptography" because it allows computation on encrypted data without revealing any information about the processed data, its applicability to general purpose computing remains limited. This is primarily the case for two reasons. First, in comparison to operations on plaintext, operations on FHE ciphertexts entail extraordinarily high memory and computation overhead [GHS12]. This is crucial because computing on plaintexts using limited computing power on trusted customer premises may be economically more efficient than computing on FHE-encrypted data using powerful resources in the cloud. Second, FHE schemes compute on encrypted data by evaluating circuits rather than executing programs as in prevailing general purpose computing. A circuit can only be evaluated completely or not at all. There is no notion of control flow which could be used to skip some computations. This in turn prevents the efficient execution of algorithms with high worst case complexity, but low average case complexity. Any execution will exhibit the same worst case behavior.

Hardware-based trusted execution environments (TEEs) such as Intel Software Guard Extensions (SGX) [Ana+13; Hoe+13; McK+13] promise to provide a secure environment for operating on encrypted data. Essentially, SGX enclave programs compute on encrypted data by decrypting the data in a protected area of the processor, performing all operations on plaintexts, and encrypting the results before they exit the processor. The security assumption is that attackers cannot observe the plaintexts in the secure processor during computation. However, it has been demonstrated that software vulnerabilities in the enclave program give attackers ample opportunity to execute arbitrary code in the enclave [Lee+17a]. These attacks can modify the control and data flow of the program and leak any secret in the program to an observer in the cloud via SGX side-channels [Bra+17; Lee+17b; Sch+17]. Since the number of software vulnerabilities grows with the size of the code base, it is advisable to keep the trusted code base (TCB) of the enclave as small as possible. Hence, it is not a good idea to outsource entire programs to an SGX enclave.

In this dissertation, we combine the described techniques for computing on encrypted data to overcome some of their limitations and drawbacks.

1.2 Contribution of this Work

The primary research question investigated in this dissertation can be stated as follows:

How can homomorphic encryption and trusted execution environments be combined into a practical architecture enabling efficient and secure computation on encrypted data?

We use *practical* to refer to the property of allowing an implementation and deployment on commercially available computer systems. With *efficient* we refer to low running time overhead in comparison to plaintext operations. Using *secure* we refer to provably secure cryptography and a small trusted code base.

Throughout this dissertation, we consider an outsourcing scenario between a trusted cloud customer (*client*) and an untrusted cloud service provider (cloud *server*). The server, however, is equipped with a *trusted (hardware) module*, which may be implemented using a TEE. The client wishes to execute a program at the cloud server with sensitive input data. The server then needs to compute on encrypted data with the assistance of the trusted module.

The efficiency of the architecture proposed in this work is based on two key ideas. The first is executing control-flow driven programs instead of computing circuits as in FHE. Doing so reveals control-flow information to the cloud server, but allows the execution of efficient algorithms without incurring the complexity penalty as in FHE. The second is performing homomorphic operations on encrypted data using multiple partially homomorphic encryption (PHE) schemes rather than FHE. Homomorphic operations of PHE schemes are much faster, but ciphertexts of one scheme are not compatible with ciphertexts of another scheme. For example, one scheme may only support homomorphic integer addition and another may only support homomorphic integer multiplication. To address the incompatibility, the trusted module is used to translate between incompatible schemes.

In our first contribution [Fis+17; Fis+20a], we study control-flow leakage under active attacks. We consider an attacker controlling the cloud server who is capable of maliciously interacting with the trusted module and altering the control and data flow of the outsourced program. We show that, if the attacker is allowed to arbitrarily interact with the trusted module, it can amplify leakage using adaptive trusted module queries. To prevent the adversary from deviating from the dataflow of the outsourced program, we introduce the concept of dataflow authentication (DFAuth). Using DFAuth, we extend the state of the art of programs executable on encrypted data to those performing control-flow decisions based on intermediate variables. DFAuth ensures only the information about the program inputs that can be inferred from the program's intended control flow are revealed to the cloud server. DFAuth computes on encrypted data using our own homomorphic authenticated symmetric encryption (HASE) scheme, which allows each control-flow decision variable to be instrumented, such that only variables with a pre-approved dataflow can be used in the decision.

In our second contribution [Fis+], we complement DFAuth and HASE with an alternative concept for operating on ciphertexts. Our trusted authenticated ciphertext operations (TACO) scheme makes use of a common authenticated symmetric encryption scheme which does not support homomorphic operations on ciphertexts. As a result, ciphertext operations have to be performed in the trusted module which hence needs to be invoked more often. However, our experiments show that the higher number of invocations is easily compensated by the use of a more efficient encryption scheme. This, in turn, allows DFAuth to be applied to applications requiring fast response times.

In our third contribution [Fis+20b], we address the concern that control-flow leakage might be inacceptable from a security perspective. To avoid simply sacrificing performance for security, we analyze the trade-off between the two in more detail. We use quantitative information flow techniques to measure leakage, running time to measure performance and program transformation techniques to alter the trade-off between the two. Combined with information flow policies, which allow developers to define varying sensitivity levels on data, we perform a policy-aware security and performance trade-off (PASAPTO) analysis. We formalize the problem of PASAPTO analysis as an optimization problem, prove the NP-hardness of the corresponding decision problem and present two algorithms solving it heuristically.

Based on the concepts of DFAuth and PASAPTO, we implemented a program transformation for Java programs. Our DFAuth trade-off analyzer (DFATA) takes Java Bytecode operating on plaintext data and an associated information flow policy as input. It outputs semantically equivalent program variants operating on encrypted data which are policy-compliant and approximately Pareto-optimal with respect to control-flow leakage and running time.

We evaluated our implementation in a commercial cloud environment using an SGX enclave as the trusted module. In our DFAuth evaluation, we for example transformed a neural network performing machine learning on sensitive medical data and a smart charging scheduler for electric vehicles (EVs). Our transformation yields a neural network with encrypted weights, which can be evaluated on encrypted inputs in 12.55 ms. Our protected EV scheduler adjusts an encrypted day-ahead charging schedule in real-time as new information arrives. Events providing new information (e.g., EV arrival, EV departure and price changes at energy markets) are processed in 1.06 s on average. In our PASAPTO evaluation, we for example applied DFATA to a decision tree program performing machine learning on medical data. The program variant with the worst performance is 357% slower than the fastest variant. Control-flow leakage varies between 0% and 17% of the input.

In comparison to a solution solely relying on SGX, our architecture provides a program-independent TCB implemented in a small trusted module, which can be reused across applications and hardened against side-channels. Thus, our approach significantly reduces the surface for attacks exploiting software vulnerabilities in the protected application. Compared to a solution solely relying on FHE, our approach provides high efficiency and actually practical performance due to fast ciphertext operations and support for control flow.

In summary, our contributions are:

- We present an architecture for computation on encrypted data based on the novel concepts DFAuth, HASE and TACO.
- We formalize the problem of PASAPTO analysis and present two heuristic algorithms approximating a solution.
- We implemented a program transformation for Java programs producing programs computing on encrypted data.
- We evaluated our implementations in a commercially available cloud environment.

1.3 Structure of this Work

The remainder of this work is structured as follows:

Chapter 2 presents foundational concepts. We introduce the notation used throughout this work, common cryptographic principles and definitions related to information theory and optimization problems.

Chapter 3 provides an overview on approaches allowing computation on encrypted data and related concepts. We discuss encryption schemes capable of computing on encrypted data, other related cryptographic primitives, techniques transforming programs such that they compute on encrypted data, and hardware-based TEEs.

Chapter 4 describes the methodology followed to investigate the research question proposed above. We first derive requirements from the research question and explain how our solution fulfills some of them by design. Then, we introduce our practicality and efficiency assessment methodology. Finally, we present our security assessment methodology.

Chapter 5 presents the foundation of our architecture and studies control-flow leakage under active attacks. We first introduce the concept of dataflow authentication (DFAuth) and show its interference equivalence property in a program dependency graph. Then, we present two constant time implementations of DFAuth: homomorphic authenticated symmetric encryption (HASE) and trusted authenticated ciphertext operations (TACO). Finally, we describe a bytecode-to-bytecode program transformation for computation on encrypted data using DFAuth. We implemented and evaluated transformed programs, for example smart charging scheduling of electric vehicles, using Intel SGX as the trusted module.

Chapter 6 addresses the concern that control-flow leakage might be inacceptable from a security perspective. We first formalize the problem of policy-aware security and performance trade-off (PASAPTO) analysis as an optimization problem and prove the NP-hardness of the corresponding decision problem. Then, we present two heuristic algorithms computing an approximation of the Pareto front of the optimization problem: a greedy heuristic providing fast convergence and a genetic algorithm providing well distributed trade-offs. We use established quantitative information flow (QIF) techniques to measure security, running time to measure performance and program transformation techniques to alter the trade-off between the two. We adjust an existing QIF analysis to capture the adversarial information flow for each variable of a program such that we can support variable-based information flow policies. We implemented our algorithms and evaluated them on programs computing on encrypted data using DFAuth in a commercially available cloud environment.

Chapter 7 concludes this dissertation. We first summarize our achievements and relate them to the primary research question. Then, we discuss open problems and directions for future work.

Reading Guide This dissertation can be read in many ways. The primary reading path is to proceed with Chapter 2 and to read through all chapters consecutively.

Chapters 5 and 6 only present new concepts and are the main chapters of this dissertation. The fundamentals for these chapters are provided in Chapters 2 and 3, and referenced from the main chapters.

Readers familiar with foundational concepts, especially cryptographic principles, may wish to skip Chapter 2 and only come back as necessary. Readers additionally knowledgeable in computation on encrypted data and trusted execution environments, especially Intel SGX, may wish to also skip Chapter 3.

Readers primarily interested in DFAuth may wish to proceed with Chapter 5 and come back to preceding chapters as necessary. Likewise, readers primarily interested in PASAPTO may wish to proceed with Chapter 6. Although PASAPTO can be applied in other contexts, we evaluate PASAPTO in the context of DFAuth and hence recommend first reading Chapter 5.

2 Preliminaries

This chapter presents foundational concepts. Section 2.1 introduces the notation used throughout this work. Section 2.2 presents common cryptographic principles used in Chapters 3 and 5. Section 2.3 provides various information theory definitions used in Chapter 6. Section 2.4 provides concepts related to optimization problems used in Chapter 6.

2.1 Notation

We use the dot notation to access object members, for example $O.A()$ refers to an invocation of algorithm A on object O. We use $:=$ for deterministic variable assignments and $=$ for comparisons. To indicate that an output of some algorithm may not be deterministic we use $\leftarrow$ instead of $:=$ in assignments.

We write $x \leftarrow_{\$} X$ to sample x uniformly at random from a set X. $|X|$ denotes the cardinality of X. For $m, n \in \mathbb{N}, m < n$ we use $[m, n]$ to refer to the set of integers $\{m, \ldots, n\}$. For a k-tuple $x = (x_1, x_2, \ldots, x_k)$ we refer to the projection of x onto its i-th ($i \in [1, k]$) component as $\pi_i(x) := x_i$. Similarly, for a set of k-tuples S we define $\pi_i(S) := \{\pi_i(x) \mid x \in S\}$.

We follow the established convention of writing the group operation of an abstract group multiplicatively. Consequently, exponentiation refers to a repetition of the group operation. We may refer to a group $(\mathbb{G}, \cdot)$ simply as $\mathbb{G}$ if the group operation is clear from the context.

Throughout the document λ denotes a security parameter and 1^λ refers to the unary encoding of λ. A function $f : \mathbb{N} \to \mathbb{R}^+$ is called *negligible* in n if for every positive polynomial p there is an n_0 such that for all $n > n_0$ it holds that $f(n) < 1/p(n)$.

To indicate that some algorithm $\mathcal{A}$ is given black-box access to some function F we write $\mathcal{A}^F$. Each parameter to F is either fixed to some variable or marked using $\cdot$ denoting that $\mathcal{A}$ may freely choose this parameter.

We denote the i-th unit vector by e_i. By $0_{m,n}$ we denote the $m \times n$ matrix with all entries equal to zero. For a vector v we write v_i to select the i-th component of v and for a matrix A we write $A_{i,:}$ to select the i-th row and $A_{:,j}$ to select the j-th column. For two matrices $A, B \in \mathbb{R}^{m \times n}$ we write $A \circ B$ to denote the Hadamard product of A and B, i.e., the componentwise multiplication.

With $s_1 \| s_2$ we denote the concatenation of bit strings s_1 and s_2.

We assume the base of the logarithmic function to be 2.

2.2 Common Cryptographic Principles

In this section, we introduce various cryptographic principles. First, we explain the concept of game-based security for precise security definitions. Then, we introduce encryption schemes providing confidentiality and message authentication codes (MACs) providing authenticity. Next, we present building blocks for MACs. Finally, we introduce authenticated encryption, a type of encryption scheme providing confidentiality and authenticity. Our presentation is based on the standard work by Katz and Lindell [KL14].

2.2.1 Game-Based Security

Modern cryptography considers security definitions consisting of a precise specification of what constitutes a break of security as well as a precise specification of the capabilities of the adversary. An established approach to provide such definitions is through *security experiments*, also called *games* [BR06]. A security experiment is performed between two probabilistic polynomial time (PPT) algorithms, a *challenger* and an *adversary* $\mathcal{A}$.

$\mathcal{A}$'s *advantage* is defined as the probability of $\mathcal{A}$ winning the game minus the probability of trivially winning the game (e.g., by guessing blindly). Security holds if all adversaries have only negligible advantage. The security proof is achieved by reducing the winning of the game to some problem that is assumed to be hard.

2.2.2 Encryption Schemes

Encryption is the process of encoding *plaintext messages* into *ciphertexts* such that only authorized parties can access them. Broadly speaking, only parties in possession of the appropriate *decryption key* are authorized and supposed to be able to restore the original message from the ciphertext. Two common types of encryption schemes are symmetric encryption and asymmetric encryption. In the following, we will formally define their syntax, correctness and security.

Symmetric Encryption

In this setting, the same key is used by the party encrypting the plaintext message and the party decrypting the ciphertext.

The classic use case involves two distinct parties who want to communicate secretly over an untrusted communication channel. Consider for example military personnel communicating via radio transmission using a key which was shared beforehand.

Another use case involves a single party who wants to communicate over an untrusted communication channel with itself over time. Consider for example the cloud storage scenario mentioned in Section 1.1. In this scenario, a cloud customer first encrypts plaintext data before transferring it to the untrusted CSP. At a later point in time, the same customer retrieves ciphertext data from the CSP and decrypts it to obtain the original plaintext data. The communication channel in this case consists of the transfer to the CSP, writing to and reading from storage, and transferring the ciphertexts back to the customer.

Definition 1 (Symmetric Encryption Syntax)**.** *A* symmetric encryption scheme *for key space $\mathcal{K}$, message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ is a triple of PPT algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *such that:*

- *The* key-generation algorithm Gen *takes as input the security parameter* 1^λ *and outputs a key $k \in \mathcal{K}$.*
- *The* encryption algorithm Enc *takes as input a key $k \in \mathcal{K}$ and a plaintext message $m \in \mathcal{M}$ and outputs a ciphertext $c \in \mathcal{C}$.*
- *The* decryption algorithm Dec *takes as input a key $k \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$ and outputs a message $m \in \mathcal{M}$.*

Definition 2 (Symmetric Encryption Correctness)**.** *Let* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a symmetric encryption scheme. We say that* Π *is* correct *if for any key k output by* $\mathsf{Gen}(1^\lambda)$ *and every $m \in \mathcal{M}$, it holds that*

$$\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m.$$

Symmetric encryption schemes are also called *private-key* encryption schemes or *secret-key* encryption schemes because the encryption key must be kept secret to achieve security. This is in contrast to *asymmetric* encryption schemes or *public-key* encryption schemes in which the encryption key may be public knowledge.

Asymmetric Encryption

In this setting, the key generation algorithm outputs a pair of keys consisting of a *public key* used for encryption and a *private key* or *secret key* used for decryption.

Definition 3 (Asymmetric Encryption Syntax)**.** *A* public-key encryption scheme *or* asymmetric encryption scheme *for message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ is a triple of PPT algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *such that:*

- *The* key-generation algorithm Gen *takes as input the security parameter* 1^λ *and outputs a pair of keys (pk, sk). We refer to pk as the* public key *and sk as the* secret key*.*
- *The* encryption algorithm Enc *takes as input a public key pk and a message $m \in \mathcal{M}$ and outputs a ciphertext $c \in \mathcal{C}$.*
- *The* decryption algorithm Dec *takes as input a secret key sk and a ciphertext $c \in \mathcal{C}$ and outputs a message $m \in \mathcal{M}$.*

Definition 4 (Asymmetric Encryption Correctness)**.** *Let* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be an asymmetric encryption scheme. We say that* Π *is* correct *if for any key pair* $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ *and every $m \in \mathcal{M}$, it holds that*

$$\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) = m$$

except with negligible probability over (pk, sk) output by $\mathsf{Gen}(1^\lambda)$*.*

Security Definitions

Numerous definitions for what constitutes a break of an encryption scheme have been proposed throughout history. For encryption schemes the security definitions of *indistinguishable encryptions under a chosen plaintext attack (IND-CPA)* and *indistinguishable encryptions under a chosen ciphertext attack (IND-CCA)* are in prevailing use. As their names suggest, the adversary goal is to distinguish between the output of encryptions in order to achieve a break and the adversary is capable of choosing plaintexts or ciphertexts.

The IND-CPA experiment for symmetric encryption proceeds as follows: First, the challenger generates a key k using $\mathsf{Gen}(1^\lambda)$. Then, the challenger invokes the adversary with input 1^λ. The adversary is given oracle access to $\mathsf{Enc}(k, \cdot)$. That means, the adversary may choose plaintexts and request their encryption from the challenger. The challenger answers each query by encrypting the requested plaintext using key k and returning the resulting ciphertext to the adversary. The adversary does not have to make all oracle queries at once, but may adaptively adjust queries depending on the ciphertexts received from the challenger. At some point, the adversary outputs a pair of plaintext messages m_0, m_1 of the same length. The challenger chooses a random bit $b \leftarrow\!\!\$\, \{0,1\}$ and encrypts the plaintext m_b as $c \leftarrow \mathsf{Enc}(k, m_b)$. Then, the challenger invokes the adversary a second time and provides the *challenge ciphertext* c as input. The adversary continues to have oracle access to $\mathsf{Enc}(k, \cdot)$. Eventually, the adversary outputs a bit b'. The output of the experiment is 1 if $b' = b$ and 0 otherwise. Since the experiment outputs 1 with probability $\frac{1}{2}$ if the adversary is guessing blindly, the advantage is defined as the probability of the experiment outputting 1 minus $\frac{1}{2}$. We now provide the full definition, including the security experiment in algorithmic notation.

Definition 5 (SE-IND-CPA)**.** *A symmetric encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *has* indistinguishable encryptions under a chosen-plaintext attack*, or is* CPA-secure*, if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\mathsf{negl}(\lambda)$ *such that:*

$$\mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{A},\Pi}(\lambda) := \Pr\left[\text{ExpSE}^{\text{IND-CPA}}_{\mathcal{A},\Pi}(\lambda) = 1\right] - \frac{1}{2} \leq \mathsf{negl}(\lambda)$$

with the experiment defined as follows:

```
$\text{ExpSE}^{\text{IND-CPA}}_{\mathcal{A},\Pi}(\lambda)$
$k \leftarrow \Pi.\mathsf{Gen}(1^\lambda)$
$(m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}^{\Pi.\mathsf{Enc}(k,\cdot)}(1^\lambda)$
$b \leftarrow\!\!\$\, \{0,1\}$
$c \leftarrow \Pi.\mathsf{Enc}(k, m_b)$
$b' \leftarrow \mathcal{A}^{\Pi.\mathsf{Enc}(k,\cdot)}(1^\lambda, c, \mathsf{st})$
**return** $b = b'$
```

Note that the state variable st is a technicality required to transport data from the first invocation of the adversary to the second invocation, because $\mathcal{A}$ is not assumed to have persistent memory between the two invocations.

The definition of IND-CPA covers many real world attack scenarios. For example, it considers the case where an attacker can obtain ciphertexts by eavesdropping on a public communication channel. It also covers the case where an attacker knows that a certain ciphertext corresponds to a certain plaintext. It even considers the case where an attacker can get the victim to encrypt chosen plaintext without the victim noticing. However, the adversary in the IND-CCA definition is even more powerful.

The IND-CCA experiment proceeds like the IND-CPA experiment and the adversary goal again is to distinguish between two ciphertexts. The key difference is that in the IND-CCA experiment the adversary has access to an additional decryption oracle. The adversary may use the decryption oracle adaptively in its first and its second invocation. However, in the second invocation the challenger refuses to provide the correct answer when asked to decrypt the challenge ciphertext c. Without this restriction, an adversary could trivially win the experiment.

We now provide the full definition of IND-CCA security for symmetric encryption. The security definitions for asymmetric encryption are identical, except that instead of providing an encryption oracle, the public key is provided to the adversary such that the adversary can perform any encryptions on its own.

Definition 6 (SE-IND-CCA)**.** *A symmetric encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *has* indistinguishable encryptions under a chosen-ciphertext attack*, or is* CCA-secure*, if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\mathsf{negl}(\lambda)$ *such that:*

$$\mathsf{Adv}^{\text{IND-CCA}}_{\mathcal{A},\Pi}(\lambda) := \Pr\left[\text{ExpSE}^{\text{IND-CCA}}_{\mathcal{A},\Pi}(\lambda) = 1\right] - \frac{1}{2} \leq \mathsf{negl}(\lambda)$$

with the experiment defined as follows:

```
$\text{ExpSE}^{\text{IND-CCA}}_{\mathcal{A},\Pi}(\lambda)$
$S := \emptyset$
$k \leftarrow \Pi.\mathsf{Gen}(1^\lambda)$
$(m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}^{\Pi.\mathsf{Enc}(k,\cdot), D(k,\cdot)}(1^\lambda)$
$b \leftarrow_\$ \{0,1\}$
$c \leftarrow \Pi.\mathsf{Enc}(k, m_b)$
$S := \{c\}$
$b' \leftarrow \mathcal{A}^{\Pi.\mathsf{Enc}(k,\cdot), D(k,\cdot)}(1^\lambda, c, \mathsf{st})$
**return** $b = b'$
```

```
$D(k, c)$
**if** $c \in S$ **then**
    **return** $\perp$
**else**
    $m := \Pi.\mathsf{Dec}(k, c)$
    **return** $m$
```

The Elgamal Encryption Scheme

A well-known example of an encryption scheme is the asymmetric scheme due to Taher Elgamal [Elg85]. The Elgamal scheme operates on a cyclic group and provides IND-CPA security under the assumption that the decisional Diffie-Hellman (DDH) problem is hard. In the following, we present the DDH problem, the construction of the Elgamal scheme and its security theorem.

Definition 7 (Group Generation Algorithm)**.** *A* group generation algorithm *is a PPT algorithm which takes 1^λ as input and outputs $(\mathbb{G}, q, g)$ where $\mathbb{G}$ is (a description of) a cyclic group, q is the order of $\mathbb{G}$ and g is a generator of $\mathbb{G}$.*

Definition 8 (Decisional Diffie-Hellman Problem)**.** *Let $\mathcal{G}$ be a group generation algorithm. We say that* the Decisional Diffie-Hellman (DDH) problem is hard relative to *$\mathcal{G}$ if for all PPT algorithms $\mathcal{A}$ there is a negligible function* $\mathsf{negl}(\lambda)$ *such that:*

$$\left| \Pr\left[\mathcal{A}(\mathbb{G}, q, g, g^\alpha, g^\beta, g^\gamma) = 1\right] - \Pr\left[\mathcal{A}(\mathbb{G}, q, g, g^\alpha, g^\beta, g^{\alpha\beta}) = 1\right] \right| \leq \mathsf{negl}(\lambda)$$

where in each case the probabilities are taken over the experiment in which $\mathcal{G}(1^\lambda)$ outputs $(\mathbb{G}, q, g)$, and then $\alpha, \beta, \gamma \leftarrow_\$ \mathbb{Z}_q$.

Construction 1 (Elgamal Asymmetric Encryption)**.** *Let $\mathcal{G}$ be a group generation algorithm. The Elgamal asymmetric encryption scheme consists of the following PPT algorithms:*

- Gen*: on input 1^λ obtain $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^\lambda)$. Choose $x \leftarrow_\$ \mathbb{Z}_q$ and compute $h := g^x$. The public key is $(\mathbb{G}, q, g, h)$, the private key is $(\mathbb{G}, q, g, x)$.*

- Enc*: on input a public key $pk = (\mathbb{G}, q, g, h)$ and a message $m \in \mathbb{G}$. Choose $r \leftarrow_\$ \mathbb{Z}_q$, compute $c_1 = g^r$ as well as $c_2 = h^r \cdot m$. Return the ciphertext $c = (c_1, c_2)$.*

- Dec*: on input a private key $sk = (\mathbb{G}, q, g, x)$ and a ciphertext $c = (c_1, c_2) \in \mathbb{G} \times \mathbb{G}$. Output the plaintext $m' = {c_1}^{-x} \cdot c_2$.*

Theorem 1 (Elgamal IND-CPA Security)**.** *Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be Construction 1 and let $\mathcal{G}$ be the group generation algorithm in* Gen*. If the DDH problem is hard relative to $\mathcal{G}$, then Π is CPA-secure.*

2.2.3 Message Authentication Codes

Encryption schemes provide confidentiality, but do not necessarily prevent ciphertexts from being manipulated without detection. This aspect is separately captured by the *authenticity* security property, which can be achieved using *message authentication codes (MACs).* In the following, we define the syntax and correctness of MACs and present the prevailing security definition.

Definition 9 (Message Authentication Code Syntax)**.** *A* message authentication code *(or MAC) is a triple of PPT algorithms* $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ *such that:*

- *The* key-generation algorithm Gen *takes as input the security parameter 1^λ and outputs a key k with $|k| \geq \lambda$.*

- *The* tag-generation algorithm Mac *takes as input a key k and a message m and outputs a* tag *$t \leftarrow \mathsf{Mac}(k, m)$.*

- *The* verification algorithm Vrfy *takes as input a key k, a message m and a tag t. It outputs a bit b, with $b = 1$ indicating that t is a valid tag for message m under k and 0 otherwise.*

Definition 10 (Message Authentication Code Correctness)**.** *Let* $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ *be a message authentication code. We say that* Π *is* correct *if for any key k and every $m \in \{0,1\}^*$, it holds that*

$$\mathsf{Vrfy}(k, m, \mathsf{Mac}(k, m)) = 1$$

except with negligible probability over k output by $\mathsf{Gen}(1^\lambda)$.

Definition 11 (MAC-UF-CMA)**.** *A message authentication code* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is* existentially unforgeable under an adaptive chosen-message attack*, or just* secure*, if for all PPT adversaries $\mathcal{A}$ there is a negligible function* $\mathsf{negl}(\lambda)$ *such that:*

$$\Pr\left[\mathrm{ExpMAC}^{\text{UF-CMA}}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

where the experiment is defined as follows:

```
$\mathrm{ExpMAC}^{\text{UF-CMA}}_{\mathcal{A},\Pi}(\lambda)$
$M := \emptyset$
$k \leftarrow \Pi.\mathsf{Gen}(1^\lambda)$
$(m, t) \leftarrow \mathcal{A}^{M(k,\cdot)}(1^\lambda)$
**return** $\mathsf{Vrfy}(k, m, t) = 1 \land m \notin M$
```

```
$M(k, m)$
$t \leftarrow \Pi.\mathsf{Mac}(k, m)$
$M := M \cup \{m\}$
**return** $t$
```

2.2.4 Building Blocks for Message Authentication Codes

Secure message authentication codes can for example be constructed from collision-resistant hash functions or pseudorandom functions. We introduce these building blocks since they can also be applied outside the scope of MACs.

Definition 12 (Hash Function)**.** *A* hash function *(with output length l) is a pair of PPT algorithms* (Gen, H) *such that:*

- Gen *takes as input the security parameter 1^λ and outputs a key s.*
- *H takes as input a key s and a string $x \in \{0,1\}^*$ and outputs a string $y \in \{0,1\}^{l(\lambda)}$ where λ corresponds to the security parameter used by* Gen *to generate s.*

Definition 13 (Collision Resistance)**.** *A hash function* $\Pi = (\mathsf{Gen}, H)$ *is called* collision-resistant *if for all PPT adversaries $\mathcal{A}$ there is a negligible function* $\mathsf{negl}(\lambda)$ *such that:*

$$\Pr[\mathrm{HashColl}_{\mathcal{A},\Pi}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

with the experiment defined as follows:

```
$\underline{\text{HashColl}_{\mathcal{A},\Pi}(\lambda)}$
$s \leftarrow \Pi.\mathsf{Gen}(1^\lambda)$
$(x, x') \leftarrow \mathcal{A}(s)$
return $x \neq x' \wedge H(s, x) = H(s, x')$
```

Definition 14 (Pseudorandom Function). *Let X and Y be two finite sets and denote the set of all functions from X to Y as $\mathcal{F}$. We say that an efficiently computable keyed function $F : \mathcal{K} \times X \to Y$ with keyspace $\mathcal{K}$ is a pseudorandom function (PRF), if for all PPT algorithms $\mathcal{A}$ there is a negligible function* $\mathsf{negl}(\lambda)$ *such that:*

$$\left|\Pr\left[\mathcal{A}^{F(k,\cdot)}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{f(\cdot)}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

where the first probability is taken over $k \leftarrow_\$ \mathcal{K}$ and the second probability is taken over $f \leftarrow_\$ \mathcal{F}$.

2.2.5 Authenticated Encryption

An authenticated encryption scheme provides both, confidentiality and authenticity. In this section, we first introduce the security definition of *unforgeability under a chosen plaintext attack (UF-CPA)*, a variant of the MAC-UF-CMA definition for encryption schemes. This definition is necessary since the syntax of a MAC is not compatible with the syntax of an encryption scheme. Then, we formally define authenticated encryption.

Definition 15 (SE-UF-CPA). *A symmetric encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is* unforgeable under a chosen-plaintext attack*, or just* unforgeable*, if for all PPT adversaries $\mathcal{A}$ there is a negligible function* $\mathsf{negl}(\lambda)$ *such that:*

$$\Pr\left[\text{ExpSE}^{\text{UF-CPA}}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

where the experiment is defined as follows:

```
$\underline{\text{ExpSE}^{\text{UF-CPA}}_{\mathcal{A},\Pi}(\lambda)}$
$M := \emptyset$
$k \leftarrow \Pi.\mathsf{Gen}(1^\lambda)$
$c \leftarrow \mathcal{A}^{E(k,\cdot)}(1^\lambda)$
$m := \Pi.\mathsf{Dec}(k, c)$
return $m \neq \perp \wedge m \notin M$

$\underline{E(k, m)}$
$c \leftarrow \Pi.\mathsf{Enc}(k, m)$
$M := M \cup \{m\}$
return $c$
```

Definition 16 (Authenticated Encryption). *A symmetric encryption scheme is an* authenticated encryption (AE) scheme*, if it is CCA-secure and unforgeable.*

2.3 Probability and Information Theory

This section introduces various information theory definitions and related definitions from probability theory [CT06].

Definition 17 (Random Variable)**.** *Let (Ω, Σ, P) be a probability space and E a measurable space. A* random variable *is a measurable function $X : \Omega \to E$. X is called a discrete random variable if the image of X is countable.*

Definition 18 (Probability Mass Function)**.** *Let $X : \Omega \to E$ be a discrete random variable on a probability space (Ω, Σ, P). Then the* probability mass function *$p_X : E \to [0,1]$ for X is defined as*

$$p_X(x) := P\left(\{\omega \in \Omega : X(\omega) = x\}\right).$$

Definition 19 (Shannon Entropy)**.** *Let X be a discrete random variable with possible values $E = \{x_1, \ldots, x_n\}$ and probability mass function $p_X(x)$. The* Shannon entropy *(or* entropy*) is defined as*

$$H(X) := -\sum_{i=1}^{n} p_X(x_i) \log p_X(x_i).$$

Definition 20 (Conditional Shannon Entropy)**.** *Let X and Y be discrete random variables with probability mass functions $p_X(x)$ and $p_Y(y)$. The* conditional Shannon entropy of X given Y *is defined as*

$$H(X|Y) := -\sum_{x \in \mathcal{X}, y \in \mathcal{Y}} p_{X,Y}(x,y) \log \frac{p_{X,Y}(x,y)}{p_Y(y)},$$

where $\mathcal{X}$ and $\mathcal{Y}$ denote the support sets of X and Y and $p_{X,Y}$ the joint probability mass function of X and Y.

Definition 21 (Conditional Minimal Entropy)**.** *Let X and Y be discrete random variables with probability mass functions $p_X(x)$ and $p_Y(y)$. The* conditional minimal entropy of X given Y *is defined as*

$$H_\infty(X|Y) := \min_{y \in \mathcal{Y}} H(X|Y = y),$$

where $\mathcal{Y}$ denotes the support set of Y.

2.4 Optimization Problems

This section introduces concepts related to multi-objective optimization problems. Section 2.4.1 provides basic definitions and Section 2.4.2 presents the hypervolume indicator for determining the quality of multi-objective approximation sets.

2.4.1 Multi-Objective Optimization Problems

The presentation in this section is based on the work by Miettinen [Mie98] and Zitzler et al. [ZBT06].

Definition 22 (Multi-Objective Optimization Problem)**.** *A* multi-objective optimization problem (MOP) *is an optimization problem of the form:*

$$\begin{array}{ll} \textit{minimize} & \{f_1(x), f_2(x), \ldots, f_k(x)\} \\ \textit{subject to} & x \in X \end{array}$$

for $k \geq 2$ objective functions *$f_i : W \to \mathbb{R}$. W is called the* decision variable space *and usually $W = \mathbb{R}^n$. X is a non-empty subset of W and called the* feasible region *or* feasible set*. An element $x \in X$ is called a* feasible solution *or* feasible decision*. The vector of objective functions*

$$f(x) := (f_1(x), f_2(x), \ldots, f_k(x))^T$$

can be interpreted as a function $f : W \to \mathbb{R}^k$ assigning each decision variable an objective vector *from the* objective space *$\mathbb{R}^k$.*

Definition 23 (Pareto Dominance)**.** *Let f be the vector of objective functions of a k-objective minimization problem with feasible set X. Furthermore, let $x, y \in X$. We say that x* weakly dominates *y, or $x \succeq y$, iff*

$$\forall i \in [1, k] : f_i(x) \leq f_i(y).$$

We say that x dominates *y, or $x \succ y$, iff*

$$x \succeq y \wedge \exists i \in [1, k] : f_i(x) < f_i(y).$$

Definition 24 (Pareto Dominance For Sets)**.** *Let X be the feasible set of a minimization problem and let $A \subseteq X$ and $B \subseteq X$. We say that A* weakly dominates *B, or $A \succeq B$, iff*

$$\forall y \in B : \exists x \in A : x \succeq y$$

and we say that A dominates *B, or $A \succ B$, iff*

$$\forall y \in B : \exists x \in A : x \succ y.$$

Definition 25 (Pareto Optimality)**.** *Let f be the vector of objective functions of a k-objective minimization problem with feasible set X and dominance relation $\succ$. A feasible solution $x \in X$ is called* Pareto-optimal with respect to f *iff*

$$\nexists y \in X : y \succ x.$$

If f is clear from context, we simply call x Pareto-optimal*. The set of all Pareto-optimal solutions is called the* Pareto front*.*

Definition 26 (Non-Dominated Set)**.** *Let f be the vector of objective functions of a k-objective minimization problem with feasible set X and dominance relation $\succ$. A set $Y \subseteq X$ is called* non-dominated *iff*

$$\nexists x, y \in Y : x \succ y.$$

2.4.2 The Hypervolume Indicator

Depending on its exact parameters, finding the Pareto front of an MOP can be computationally expensive or prohibitive. In this case, a common practice is to instead compute an approximation of the Pareto front using an MOP optimizer or an application-specific heuristic. These algorithms then output a set of objective vectors as an *approximation set*. Given an MOP with objective space Z, an approximation set is an element of the powerset of Z. In almost all cases, only feasible solutions are accepted and the feasible set X is used instead of Z.

Clearly, the Pareto front is the best approximation set that can be found. But, how can we determine whether an approximation set is better than another? An established measure used to determine the quality of approximation sets is the hypervolume indicator (HVI) [ZBT06; ZT98; RVLB15; Aud+18]. Essentially, an indicator assigns a numeric quality score to an approximation set. Geometrically, the HVI computes the space (or volume) dominated by a given approximation set.

Classically, the HVI has been defined using volumes of polytopes or hypercubes [ZBT06]. Zitzler et al. [ZBT06] provide an elegant definition using attainment functions, which we present in the following. Their definition assumes $Z = (0,1)^k$, but this assumption is without loss of generality since there exists a bijective mapping from $\mathbb{R}$ into the open interval $(0,1) \subset \mathbb{R}$ [ZBT06]. Their definitions use the weak dominance relation $\succeq$, but likewise applies to the dominance relation $\succ$.

Definition 27 (Attainment Function)**.** *Given an approximation set A, the attainment function $\alpha_A : [0,1]^n \to \{0,1\}$ for A is defined as*

$$\alpha_A(z) := \begin{cases} 1 & \textit{if } A \succeq \{z\} \\ 0 & \textit{else} \end{cases}$$

for $z \in Z$.

Definition 28 (Hypervolume Indicator)**.** *The* hypervolume indicator (HVI) I_H^* *with reference point $(0,\dots,0)$ is defined as*

$$I_H^*(A) := \int_{(0,\dots,0)}^{(1,\dots,1)} \alpha_A(z)dz$$

for an approximation set A.

3 Related Work

This chapter provides an overview on approaches allowing computation on encrypted data and related concepts. We discuss encryption schemes capable of computing on encrypted data in Section 3.1 and other cryptographic primitives in Section 3.2. Section 3.3 reviews techniques transforming programs to compute on encrypted data. We close the chapter with a discussion of hardware-based trusted execution environments in Section 3.4. Note that additional related work is presented in Sections 5.8 and 6.8.

3.1 Encryption Schemes with Computation Support

This section presents encryption schemes supporting computation on ciphertexts. Sections 3.1.1 and 3.1.2 introduce property-preserving and searchable encryption. Sections 3.1.3 and 3.1.4 discuss functional and homomorphic encryption.

3.1.1 Property-Preserving Encryption

Essentially, a property-preserving encryption scheme is a symmetric encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ extended with a fourth Test algorithm, which can be applied to a tuple of ciphertexts in order to determine whether their corresponding plaintexts possess a certain pre-defined property. In the formal definition provided by Pandey et al. [PR12] the Gen algorithm is called Setup and, in addition to a secret key, also outputs some public parameters which are provided to all other algorithms. In the following, we present their definition using our notation.

Definition 29 (Property-Preserving Encryption Syntax)**.** *A* symmetric property-preserving encryption (PPE) *scheme for message space $\mathcal{M}$ is a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Test})$ *and an associated property $P : \mathcal{M}^l \to \{0,1\}$ such that:*

- *The* setup algorithm Setup *takes as input the security parameter λ and outputs a secret key k and public parameters pp.*
- *The* encryption algorithm Enc *takes as input public parameters pp, a key k and a plaintext message $m \in \mathcal{M}$ and outputs a ciphertext c.*
- *The* decryption algorithm Dec *takes as input public parameters pp, a key k and a ciphertext c and outputs a message $m \in \mathcal{M}$.*
- *The* testing algorithm Test *takes as input public parameters pp, ciphertexts $c_1, \ldots, c_l$ and outputs a bit $b \in \{0,1\}$.*

Definition 30 (Property-Preserving Encryption Correctness)**.** *Let* $\Pi = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Test})$ *be a PPE scheme for message space* $\mathcal{M}$ *with associated property* P*. In order for* Π *to be correct, we require that for any* $(pp, k) \leftarrow \mathsf{Setup}(1^\lambda)$ *and any* $m \in \mathcal{M}$ *it holds that*

$$\mathsf{Dec}(pp, k, \mathsf{Enc}(pp, k, m)) = m.$$

Furthermore, it is required that there exists a negligible function $\mathsf{negl}(\lambda)$ *such that* $\forall (m_1, \ldots, m_l) \in \mathcal{M}^l$

$$\Pr\left[\begin{array}{l} \mathsf{Test}(pp, c_1, \ldots, c_l) \\ = P(m_1, m_2, \ldots, m_l) \end{array} \middle| \begin{array}{l} (pp, k) \leftarrow \mathsf{Setup}(1^\lambda) \\ \forall i \in [1, l] : c_i \leftarrow \mathsf{Enc}(pp, k, m_i) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

where the probability is taken over the randomness of all algorithms.

Since the Test algorithm of a PPE scheme does not require any secret inputs, it can be evaluated on ciphertexts by an untrusted party, for example a cloud service provider in our scenario.

Deterministic encryption schemes such as synthetic initialization vector authenticated encryption (SIV) [Har08] allow to determine whether plaintexts are equal based on their corresponding ciphertexts. Similarly, order-preserving encryption (OPE) as defined by Agrawal et al. [Agr+04] and order-revealing encryption (ORE) as defined by Boneh et al. [Bon+15] allow to determine plaintext order (e.g., $\leq$ on $\mathbb{Z}$) from ciphertexts.

Some PPE schemes, for example SIV and OPE, possess the additional feature of allowing the same Test algorithm to be used for testing the property P on plaintexts and testing the property on ciphertexts. This feature is for example useful when integrating encryption algorithms into existing data processing environments such as database management systems.

3.1.2 Searchable Encryption

Searchable symmetric encryption (SSE) as introduced by Song et al. [SWP00] allows to test for equality similar to deterministic encryption. However, basically, the PPE Test algorithm cannot be applied to all ciphertexts, but first requires a search token to be generated.

Definition 31 (Searchable Symmetric Encryption Syntax)**.** *A* searchable symmetric encryption (SSE) *scheme for message space* $\mathcal{M}$ *is a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Match})$ *such that:*

- *The* setup algorithm Setup *takes as input the security parameter* λ *and outputs a secret master key* k*.*
- *The* encryption algorithm Enc *takes as input a master key* k *and a plaintext message* $m \in \mathcal{M}$*. It outputs a ciphertext* c_m*.*
- *The deterministic* token generation algorithm Token *takes as input a master key* k *and a plaintext message* $w \in \mathcal{M}$*. It outputs a search token* t_w*.*

- *The deterministic* testing algorithm Match *takes as input a ciphertext* c_m *and a search token* t_w*. It outputs a bit* $b \in \{0, 1\}$*.*

Definition 32 (Searchable Symmetric Encryption Correctness)**.** *Let* $\Pi = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Match})$ *be an SSE scheme for message space* $\mathcal{M}$*. In order for* Π *to be correct, we require that*

$$\Pr\left[\begin{array}{c|c} \mathsf{Match}(c_m, t_w) & k \leftarrow \mathsf{Setup}(1^\lambda), \forall m, w \in \mathcal{M} : \\ = (m = w) & c_m \leftarrow \mathsf{Enc}(k, m), t_w := \mathsf{Token}(k, w) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

where the probability is taken over the randomness of all algorithms.

In particular, Song et al. [SWP00] consider a collection of text documents outsourced to a CSP. Using SSE, a trusted client can perform exact keyword matching on the outsourced documents by first generating a search token using the master key and then sending the token to the CSP. The CSP can then perform the matching on the documents, but is limited to learning matches of keywords for which a token was provided.

Note that the syntax definition does not include a decryption algorithm. If decryption is required, an SSE scheme can be combined with a regular symmetric encryption scheme, e.g., an authenticated encryption scheme.

Boneh et al. [Bon+04] later proposed public-key encryption with keyword search (PEKS), which allows any party to encrypt using the public key, but limits the generation of search tokens to entities in possession of the secret master key. For an overview on additional variants of searchable encryption, e.g., range queries rather than equality testing, we refer to a survey by Bösch et al. [Bös+14].

3.1.3 Functional Encryption

Functional encryption [BSW11] generalizes the concept of searchable encryption in the asymmetric setting from equality testing to function evaluation. As usual in asymmetric encryption, the encryption algorithm takes a public key and a plaintext message, and outputs a ciphertext. However, the decryption algorithm does not only reverse the encryption process, it evaluates a function at the same time. The function is identified by a function-specific secret key generated from a master key. Constructions have been proposed in which the function key corresponds to a binary circuit [Gar+16; Gol+13b] or Turing machine [Gol+13a].

Definition 33 (Functional Encryption Syntax)**.** *A* functional encryption (FE) *scheme for key space* $\mathcal{K}$*, message space* $\mathcal{M}$ *and* functionality $F : \mathcal{K} \times \mathcal{M} \to \{0, 1\}^*$ *is a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *such that:*

- *The* setup algorithm Setup *takes as input the security parameter* λ*. It outputs a secret master key* mk *and a public key* pk*.*
- *The* key-generation algorithm Gen *takes as input a master key* mk *and function key* k*. It outputs a secret key* sk_k *for evaluating* $F(k, \cdot)$*.*
- *The* encryption algorithm Enc *takes as input a public key* pk *and a plaintext message* $m \in \mathcal{M}$*. It outputs a ciphertext* c_m*.*

- *The* decryption algorithm Dec *takes as input a secret key sk_k and a ciphertext c_m. It outputs the result of $F(k, m)$.*

Definition 34 (Functional Encryption Correctness)**.** *Let $\Pi = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an FE scheme for key space $\mathcal{K}$, message space $\mathcal{M}$ and functionality $F : \mathcal{K} \times \mathcal{M} \to \{0, 1\}^*$. In order for Π to be correct, it is required that*

$$\begin{aligned}
&\forall k \in \mathcal{K}, m \in \mathcal{M},\\
&\forall (pk, mk) \leftarrow \mathsf{Setup}(1^\lambda)\\
&\forall sk \leftarrow \mathsf{Gen}(mk, k)\\
&\forall c \leftarrow \mathsf{Enc}(pk, m)
\end{aligned}$$

it holds that

$$\mathsf{Dec}(sk, c) = F(k, m).$$

PEKS [Bon+04] can be considered a special case of FE for exact keyword matching. Functionality provided by similarly limited schemes for example include inner-product computation and orthogonality testing of encrypted vectors [AFV11; Lew+10].

3.1.4 Homomorphic Encryption

We already briefly introduced and discussed the concept of homomorphic encryption (HE) in Chapter 1. In comparison to functional encryption, function evaluation in HE does not depend on a secret key and always yields a ciphertext; that is, the result stays encrypted. HE schemes can be categorized based on the limitations imposed onto the functions that can be evaluated homomorphically.

A *partially homomorphic encryption (PHE)* scheme is limited to a single operation on ciphertexts, e.g., integer addition or integer multiplication, but the number of such operations is not limited. We provide a formal definition in the asymmetric encryption setting based on the definition by Katz and Lindell [KL14]. Equivalently, homomorphic encryption could be defined using a fourth Eval algorithm.

Definition 35 (Asymmetric Homomorphic Encryption)**.** *An asymmetric encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a* partially homomorphic encryption (PHE) *scheme, or just* homomorphic*, if for all λ and all $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ it is possible to define groups $(\mathbb{M}, +)$ and $(\mathbb{C}, \cdot)$ (depending on pk only) such that:*

- *The message space is $\mathbb{M}$, and all ciphertexts output by $\mathsf{Enc}(pk, \cdot)$ are elements of $\mathbb{C}$.*
- *For any $m_1, m_2 \in \mathbb{M}$, any $c_1 \leftarrow \mathsf{Enc}(pk, m_1)$, and any $c_2 \leftarrow \mathsf{Enc}(pk, m_2)$ it holds that:*

  $$\mathsf{Dec}(sk, c_1 \cdot c_2) = m_1 + m_2.$$

  Moreover, the distribution on ciphertexts obtained by encrypting m_1, encrypting m_2, and then multiplying the results is identical *to the distribution on ciphertexts obtained by encrypting $m_1 + m_2$.*

PHE schemes have for example been proposed by Paillier [Pai99b] for homomorphic addition on integers and Goldwasser et al. [GM82] for homomorphic computation of the bitwise exclusive-or (XOR). We also already presented a construction of a PHE scheme. The Elgamal encryption scheme provided in Construction 1 can homomorphically compute the group operation with $\mathbb{M} = \mathbb{G}$ and $\mathbb{C} = \mathbb{G} \times \mathbb{G}$, where the group operation in $\mathbb{C}$ is performed component-wise.

The BGN encryption scheme by Boneh et al. [BGN05] is a well-known representative of the more powerful category of *somewhat homomorphic encryption (SWHE)* schemes. Besides an arbitrary number of additions on ciphertexts, their scheme supports up to one homomorphic multiplication. Thus, BGN can be used to homomorphically evaluate boolean circuits in 2-DNF form, i.e., disjunctive normal form where each conjunctive clause contains at most two literals.

Extending on SWHE schemes, Gentry presented the first construction of a *fully homomorphic encryption (FHE)* scheme in 2009 [Gen09]. An FHE scheme can perform an unlimited number of homomorphic additions and an unlimited number of homomorphic multiplications. As a result, an FHE scheme can be used to homomorphically evaluate arbitrary boolean circuits and thus can perform arbitrary computations on encrypted data.

Although Gentry's discovery sparked significant interest in the field which led to a number of additional research results [BGV14; BV14; SV10; Dij+10], the applicability of FHE to general purpose computing — and in particular cloud computing — remains limited. The reason for this is twofold.

First, in comparison to operations on plaintext, operations on FHE ciphertexts entail extraordinarily high memory and computation overhead. This is crucial for cloud applications because computing on plaintexts using limited resources in a trusted environment may be economically more efficient than computing on FHE-encrypted data using powerful resources in the cloud. An often used benchmark for FHE schemes is the homomorphic evaluation of an AES-128 encryption operation, which Gentry et al. [GHS12] report to require about 4 minutes of wall-clock time and 3 GB of random access memory (RAM).

Second, there is an important difference between evaluating circuits and executing programs as in prevailing general purpose computing. A circuit can only be evaluated completely or not at all. There is no notion of control flow which could be used to skip some computations. This in turn prevents efficient execution of algorithms with high worst case complexity, but low average case complexity. Any execution will exhibit the same worst case behavior, since circuits must be evaluated entirely.

3.2 Other Cryptographic Primitives

In this section, we briefly discuss cryptographic primitives related to computation on encrypted data but outside the scope of this dissertation. Section 3.2.1 introduces secure multi-party computation protocols. Section 3.2.2 presents primitives for hiding the access pattern when handling collections of ciphertexts. Section 3.2.3 discusses cryptographic program code obfuscation.

3.2.1 Secure Multi-Party Computation

Secure multi-party computation (MPC) [EKR18] allows multiple distrusting parties to jointly compute an arbitrary function while keeping their inputs secret. The security goal is to ensure that each party learns nothing beyond their intended output function.

MPC is best illustrated using Yao's Millionaires' problem [Yao82] which considers two millionaires, who want to know which of them is richer without revealing their actual wealth. Assuming their respective wealths are w_a and w_b, then each of the two wants to learn the output function $w_a \leq w_b$ without revealing any other information to the other party.

Secure computations are performed using MPC protocols in an interactive setting in which all parties are assumed connected via a network and online. A primary complexity measure for protocols then is the number of communication rounds required for a given computation.

MPC was first introduced for two parties [Yao82] and later extended to more parties [GMW87]. An extension has been proposed in which the parties are assisted by an additional server [FKN94]. The server has no inputs to the secure computation and learns no output function, but provides its computing resources to the other parties.

MPC is related to the setting considered in this dissertation, but does not directly apply. Whereas MPC considers multiple mutually distrusting parties, we consider a trusted client and an untrusted server. The server trusts the client and has no inputs that must be kept secret from the client.

3.2.2 Primitives for Access Pattern Hiding

Encryption can be used to transform plaintexts into ciphertexts to achieve data confidentiality. However, when multiple ciphertexts are held in data structures allowing ciphertexts to be accessed individually, additional security properties may become relevant.

An adversary capable of observing accesses to individual ciphertexts of a collection may be able to infer information about their corresponding plaintexts based on the observed access pattern. Consider for example a cloud storage scenario in which an array of ciphertexts is outsourced to a CSP. An eavesdropping attacker on the cloud server could observe which array elements are accessed, the order of accesses and whether an access performed a read or a write operation.

Oblivious RAM (ORAM) is a cryptographic primitive allowing a trusted client to access a random access memory (RAM) outsourced to an untrusted server without revealing the access pattern to the server. Formally, given a readable and writable RAM at the server, ORAM (in combination with encryption) at the client ensures that an adversary cannot distinguish between the RAM communication transcripts for two access sequences of the same length, even if the sequences were chosen by the adversary. ORAM constructions have been proposed for various assumptions about available client storage, server computation capabilities and number of servers [Abr+17; Dev+16; Ren+15; Ste+13].

Private information retrieval (PIR) [KO97] considers the similar problem of hiding which entries a client has requested from a read-only RAM outsourced to a server. In comparison to ORAM, PIR considers multiple clients and the data held by the server is assumed to be public. That is, the data does not have to be kept confidential from the server.

3.2.3 Obfuscation Primitives

Colloquially, program obfuscation is the process of producing program code that is difficult for humans to understand. Software developers may for example want to use obfuscation to hide the true purpose of a program or to prevent reverse engineering. In the scenario considered in this dissertation, the client may want to prevent the CSP from learning the source code of the outsourced program.

Ideally, an obfuscator $\mathcal{O}$ efficiently transforms a given program P into a program $P' = \mathcal{O}(P)$ such that P' computes the same function as P and anything that can be efficiently computed from P' can be efficiently computed given only oracle access to P, that is, P' behaves like a virtual black box for P.

Barak et al. [Bar+12] show that it is impossible to achieve this notion of virtual black box obfuscation for arbitrary programs. As an alternative security notion, the authors introduce the weaker definition of indistinguishability obfuscation (iO) in which the black box paradigm is avoided. In a nutshell, this notion requires that, if two programs compute the same function, then their obfuscations should be indistinguishable. Garg et al. [Gar+13] proposed a possible construction for iO. However, their work applies to circuits as used in FHE rather than programs as considered in this work.

Due to ongoing research in the field and without any readily available and efficient cryptographic tools to use, the architecture presented in this dissertation does not consider the protection of outsourced program code.

3.3 Program Transformation Techniques

Since fully homomorphic encryption entails high computational overhead, researchers have resorted to partially encrypting computations using more efficient encryption schemes. Several proposals for program transformation into such encrypted computations have been made.

MrCrypt [Tet+13] uses type inference to infer feasible encryption schemes for each program variable and transforms programs such that they operate using the inferred encryption schemes. If no arithmetic or comparison operation is performed, MrCrypt uses a common symmetric encryption scheme providing IND-CPA security. Additionally, MrCrypt may infer deterministic encryption for equality comparisons and OPE for order comparisons. Multiplication and addition operations are supported using two PHE schemes. MrCrypt may also infer that FHE is required for the execution of a program, but the authors did not incorporate an FHE implementation due to the high computational overhead. Essentially, MrCrypt attempts to partition a program such that it can run using a set of efficient encryption schemes. However, MrCrypt does not consider conversions between incompatible encryption schemes. If FHE is inferred, then no efficient partition could be found. This severely limits the set of programs that can be transformed using MrCrypt. The authors evaluate MrCrypt on shallow Java MapReduce programs and even in this case several test cases cannot be executed.

JCrypt [DMD16] improves the type inference algorithm such that a larger set of programs is admissible for computation on encrypted data. By distinguishing between sensitive and cleartext variables JCrypt increases the granularity of the program partition. Additionally, JCrypt allows instructions of the original program to execute in the sensitive or the cleartext context, depending on whether there is a data flow from a variable marked as sensitive. Thus, the use of encryption is minimized and the set of admissible programs increased. However, still no conversions between encryption schemes are considered.

AutoCrypt [Top+13] uses partially homomorphic encryption for addition and multiplication, searchable symmetric encryption for equality comparisons, but decided against using OPE due to its weak security guarantees. The AutoCrypt architecture assumes an untrusted virtual machine running programs on encrypted data on top of a trusted hypervisor. The hypervisor can be used to convert between incompatible encryption schemes. However, the authors realized the security implications of allowing arbitrary conversions. Hence, they disallowed any conversion from homomorphic encryption to searchable encryption. This restriction prevents any program from running that modifies its input and then performs a control-flow decision.

Next to programs written in imperative languages (e.g., Java) programs in declarative languages (e.g., SQL) are amenable to encrypted computation. In these languages, the programmer does not specify the control-flow decisions, but they may be optimized by the interpreter or compiler. Hence any resulting data is admissible and weaker encryption schemes must be used. Hacigümüş et al. [Hac+02] used deterministic encryption to implement a large subset of SQL. Popa et al. [Pop+11] additionally used randomized and order-preserving encryption in an adjustable manner.

3.4 Trusted Execution Environments

Another approach for computing on encrypted data are *trusted execution environments (TEEs)*. Instead of relying on cryptographic guarantees derived from theoretical hardness assumptions, TEEs base their security on practical hardware assumptions.

The general concept of hardware-based security is in prevailing use today. For example, smart cards are used for access control and authorization in mobile telephony networks and banking. These mechanisms are ultimately implemented by using the card's processor to perform cryptographic operations such as encryption or MAC computation (cf. Section 2.2). The security assumption is that the chip card is tamper-resistant, i.e., it is assumed impossible (or at least uneconomical) for an attacker to extract the cryptographic key material stored in the card's persistent memory, e.g., in order to steal the card holders's identity.

Whereas smart cards posses only constrained computation power and only provide limited functionality (e.g., cryptographic operations), TEEs aim to extend the concept of hardware-based security to the execution of application-specific code in general purpose computing. The prevailing TEE for general purpose cloud computing readily available at the time the research for this dissertation was started is Intel SGX [Ana+13; Hoe+13; McK+13]. We introduce details about SGX in Section 3.4.1 and present attacks on SGX in Section 3.4.2.

3.4.1 Intel Software Guard Extensions (SGX)

Intel Software Guard Extensions (SGX) is an instruction set extension of commercial off-the-shelf processors providing TEE capabilities. It was introduced into the Intel Core CPU family with the Skylake generation released in Q3/2015 and into the Intel Xeon family with the Kaby Lake generation released in Q1/2017.

Essentially, SGX enables developers to execute programs in isolated memory regions, so-called *enclaves*, such that the program's code and data are protected from attackers controlling the host system. Potential attackers for example include external intruders, regular applications and other enclaves running on the same system, the host operating system or hypervisor, and device firmware.

In more detail, SGX combines a number of techniques to achieve its protections. When a memory write operation is performed by an enclave program, the corresponding data is encrypted before being written to physical memory. Similarly, when a memory read operation is performed, data is decrypted before being loaded into the CPU cache. The encryption process does not only provide confidentiality of memory data, but also ensures its integrity (cf. authenticated encryption in Section 2.2.3). In addition, SGX ensures the property of *freshness*, which guarantees that the latest version of a memory page is loaded. Overall, SGX's memory encryption thus prevents a physical memory attacker from reading data as well as modifying or rolling back data without being detected. Enclave data in plaintext is only available inside the processor and the security assumption is that attackers cannot observe the plaintexts during computation. Unfortunately, the total amount of memory for which these strong guarantees are provided is limited to 128 MB. 96 MB are available to user-provided enclaves, the remainder is used for internal SGX purposes. The limitation can be overcome by swapping out some (encrypted) memory pages to the host operating system, but doing so has performance as well as security implications.

SGX applications are developed using a software development kit (SDK) and follow a certain framework. An application consists of two parts: an untrusted program part and a trusted enclave part. The untrusted part is executed as a regular operating system process (i.e., in unprotected memory) and is responsible for setting up the trusted enclave. It can interact with its environment as usual, for example access the filesystem on disk or initiate network connections. The enclave is executed in isolated memory with the protections described above. It is incapable of performing input/output operations directly, but has to use the untrusted part to do so. The two parts can communicate using an explicit interface specified at compile time. Defined *enclave calls (ECalls)* allow the untrusted part to perform well-defined calls into the enclave and *outside calls (OCalls)* allow the enclave to perform calls into the untrusted program part. Additionally, the enclave has access to the virtual memory allocated to the corresponding untrusted part. However, since this memory is not subject to the strong protections of enclave memory, security implications when reading or writing data must be taken into account.

Another feature provided by SGX, which makes it especially appealing for remote computation scenarios such as the one considered in this dissertation, is *remote attestation* [Ana+13]. In a nutshell, SGX can prove to a remote party that an enclave was initialised with a certain version of code and data. Internally, this is realized by hashing (cf. Definition 13) each memory page as it is loaded and then computing a single summary hash before completing enclave initialisation. During the attestation process, the remote party and the enclave can also establish a secure communication channel, for example by exchanging cryptographic key material. In summary, remote attestation allows the remote party to determine whether their expected version of an enclave is running before transmitting sensitive data to the remote enclave.

3.4.2 Attacks on Intel SGX

Various attacks against Intel SGX have been published in the literature. SGX has for example been demonstrated to be prone to information leakage via software side-channels, vulnerable to exploitation of security defects in enclave programs, and affected by processor bugs.

Side-Channel Attacks When memory paging is used, SGX inherently leaks information about page accesses to the host operating system. This is because SGX delegates paging to the untrusted host, which can thus learn the memory access pattern (cf. Section 3.2.2) of enclaves at the granularity of pages. Xu et al. [XCP15] show that an untrusted operating system can exfiltrate secrets from an enclave via this side-channel by inducing page faults.

Another source of information leakage are caches shared between SGX enclaves and untrusted software. Moghimi et al. [MIE17] show that cryptographic key material can be exfiltrated from a shared L1 cache. The attacks presented by Brasser et al. [Bra+17] and Götzfried et al. [Göt+17] achieve the same goal, but do not require the victim enclave to be frequently interrupted by the host operating system. Schwarz et al. [Sch+17] show how to conceal cache attacks when attacking the victim enclave from another enclave.

Yet another source of information leakage is the branch prediction history of modern CPUs. Lee et al. [Lee+17b] present a branch shadowing attack allowing the control flow executed inside SGX enclaves to be inferred from an untrusted program. The authors use this attack to recover cryptographic key material used by the victim enclave.

It should be noted that these attacks are outside of Intel SGX's threat model. SGX "is not designed to handle side-channel attacks" [Int19] and it is up to the developers to build their enclaves accordingly.

Enclave Code Exploits Weichbrodt et al. [Wei+16] present AsyncShock, a tool for the exploitation of synchronization bugs in multi-threaded enclave programs. AsyncShock interrupts threads by forcing segmentation faults on enclave pages and exploits use-after-free and time-of-check-to-time-of-use defects to hijack the control flow of the enclave program.

Lee et al. [Lee+17a] apply the exploit technique of return-oriented programming (ROP) to SGX enclave programs. ROP works by piecing together programs from code snippets preceding return statements in the victim program. The authors present Dark-ROP, which uses fuzzing on the victim enclave's ECalls to find memory corruption vulnerabilities in the enclave program. By exploiting these vulnerabilities, Dark-ROP achieves arbitrary code execution in the enclave. This in turn allows enclave code and data to be exfiltrated via untrusted memory and remote attestation to be defeated.

Since the enclave code is controlled by the enclave developer, these attacks are also outside the threat model of Intel SGX. Because the number of software vulnerabilities grows with the size of the code base, it is advisable to keep the TCB of the enclave program as small as possible.

Processor Bugs Another source of attacks has been the speculative out-of-order execution paradigm implemented in Intel CPUs. Out-of-order execution refers to the reordering of instructions to minimize CPU idle time. The idea is to execute instructions as soon as their operands are available but to make their results visible in the original order specified by the programmer. Similarly, speculative execution predicts the outcome of branching decisions and executes instructions along the predicted control-flow path before knowing whether it is actually taken or not. Both techniques entail the computation of intermediate results which must be discarded rather than become visible to the programmer.

Outside the context of SGX, Lipp et al. [Lip+18] show an attack named Meltdown exploiting intermediate results from out-of-order execution to subvert memory isolation between processes. The Spectre attack proposed by Kocher et al. [Koc+19] achieves the same result by exploiting speculative execution.

SGXPectre [Che+19] is a variant of the Spectre attack against SGX enclaves used to extract register values and cryptographic key material from enclaves. Foreshadow [Bul+18], ZombieLoad [Sch+19], CacheOut [Sch+20] and SGAxe [Sch+] are attacks similar to Meltdown capable of extracting sensitive data from enclaves and disarming the security protections of SGX (e.g., attestation).

All mentioned attacks are within the scope of the Intel SGX threat model and have been addressed by CPU microcode updates published by the processor vendor.

4 Methodology

This chapter derives requirements from our research question and summarizes the methodology used to assess them. Section 4.1 defines requirements and Section 4.2 explains how our solution fulfills some of them by design. Section 4.3 presents our practicality and efficiency assessment methodology. Section 4.4 introduces our security assessment methodology.

4.1 Solution Requirements

Section 1.2 proposes the research question investigated in this dissertation:

> How can homomorphic encryption and trusted execution environments be combined into a practical architecture enabling efficient and secure computation on encrypted data?

The same section briefly expands on parts of the question:

> We use *practical* to refer to the property of allowing an implementation and deployment on commercially available computer systems. With *efficient* we refer to low running time overhead in comparison to plaintext operations. Using *secure* we refer to provably secure cryptography and a small trusted code base.

From the first quote, we can directly derive the following initial requirements:

- R1: The provided solution must use homomorphic encryption.
- R2: The provided solution must use a trusted execution environment.
- R3: The provided solution must offer an architecture sufficiently generic to support a multitude of applications.

From the second quote, we can derive the following additional requirements:

- R4: The solution must be implementable and deployable on a commercially available computer system.
- R5: The solution must have low running time overhead in comparison to plaintext operations.
- R6: The solution must provide provable security guarantees.
- R7: The solution must be implementable using a small trusted code base.

In the remainder of this chapter, we describe how the solution provided in this dissertation fulfills these requirements by design or how we assess them.

4.2 Solution Design

Chapters 5 and 6 present a novel architecture for computation on encrypted data as an answer to the research question proposed in this dissertation. In this section, we explain how our architecture is designed to fulfill some of the requirements listed above.

Fully homomorphic encryption (cf. Section 3.1.4) can perform arbitrary computations on encrypted data in theory, but in practice suffers a complexity penalty and high computational overhead. To avoid the complexity penalty, our architecture executes control-flow driven programs instead of computing circuits as in FHE (R5). To avoid the high computational overhead, our architecture uses multiple partially homomorphic encryption schemes rather than FHE (R1, R5).

TEEs (cf. Section 3.4) can very efficiently compute on encrypted data, but have been shown to be prone to attacks exploiting software vulnerabilities in their TCB. Our architecture significantly reduces the surface for such attacks by keeping the TCB small. Instead of executing entire applications in the TEE, we consider a generic trusted module implemented using a TEE (R2). The TCB of the trusted module is small and program-independent, i.e., has a constant size (R7). As such it can be reused across applications and hardened against software vulnerabilities as well as side-channels.

4.3 Practicality and Efficiency Assessment Methodology

We demonstrate the practicality of our contributions by implementing research prototypes in software (R4). All our implementations use an Intel SGX enclave (cf. Section 3.4.1) as the TEE (R2). The SGX instruction set is widely supported by common Intel desktop and server CPUs (R4). Initial software prototypes are deployed to an off the shelf desktop computer system featuring a common Intel desktop CPU (R4). Subsequent implementations are evaluated on the Microsoft Azure platform, a commercial cloud environment providing access to SGX-capable CPUs (R4).

To assess the practical efficiency of our contributions, we evaluate our prototypes in multiple applications (R3). In Chapter 5, we evaluate the wall-clock running times of different programs executed on encrypted data and compare them to the running times of equivalent programs operating on plaintext data (R5). We also collect the number of TEE invocations required by two of our proposed primitives, such that we can also compare this dimension. In Chapter 6, the evaluation of our contributions focusses on the quality of the solutions discovered by our algorithms, rather than their performance. However, we can estimate the actual wall-clock running times of our algorithms from data collected during an exhaustive search of the solution space. In the same chapter, we also evaluate the wall-clock running times of different variants of the same program computing on encrypted data as part of our security and performance trade-off analysis (R3).

4.4 Security Assessment Methodology

Chapter 5 introduces the foundation of our architecture, including novel cryptographic primitives. To provide precise security definitions, we follow the principle of game-based security (cf. Section 2.2.1). For a new primitive, we first formally define its syntax requirements and correctness conditions. Then, we formally define its desired security properties using security experiments (games). Next, we provide one or more constructions adhering to the syntax requirements. Finally, we show the security of our constructions via reduction to problems assumed hard (R6).

We perform security reductions using a *sequence of games* [BR06]. The first game is the original security experiment provided by the security definition. Each subsequent game is equal to the previous game except for some small well-defined change for which we argue that it does only negligibly influence adversarial advantage. The last game then has a special and easy to verify property, e.g., the adversary has no advantage over a blind guess. Only negligible change in advantage between subsequent games implies only negligible change in advantage between the first and the last game, which concludes the reduction.

Chapter 6 extends the foundation of our architecture with additional security guarantees. As part of our security and performance trade-off analysis, we evaluate the leakage of different variants of the same program computing on encrypted data using the technique of quantitative information flow. QIF uses established definitions from information theory (cf. Section 2.3) to summarize the leakage of a program in a single numeric value.

5 DFAuth: Dataflow Authentication

In this chapter, we present the foundation of our architecture. We study control-flow leakage under active attacks and introduce the concept of dataflow authentication (DFAuth) as a defense.

Section 5.1 briefly reiterates our motivation, introduces a possible dataflow modification attack, and provides an overview of our solution. Section 5.2 provides our adversary model and defines the syntax, correctness and security of our homomorphic authenticated symmetric encryption (HASE) scheme. Based on HASE, we introduce DFAuth and the security it provides in Section 5.3. Section 5.4 presents our HASE constructions and discusses their security. We complement DFAuth and HASE with our alternative construction trusted authenticated ciphertext operations (TACO) in Section 5.5. Details about our implementation are given in Section 5.6. Section 5.7 shows the results of our experiments using this implementation. Section 5.8 presents related work and Section 5.9 provides a summary of this chapter.

The content of this chapter has been the subject of the following scientific publications, of which parts are included verbatim in this thesis.

- *Andreas Fischer, Benny Fuhry, Florian Kerschbaum, Eric Bodden: Computation on Encrypted Data using Dataflow Authentication. In 20th Privacy Enhancing Technologies Symposium (PETS), 2020.* [Fis+20a]
- *Andreas Fischer, Benny Fuhry, Jörn Kussmaul, Jonas Janneck, Florian Kerschbaum, Eric Bodden: Computation on Encrypted Data using Dataflow Authentication. Under submission.* [Fis+]

5.1 Introduction

Many critical computations are being outsourced to the cloud. However, attackers might gain control of the cloud servers and steal the data they hold. End-to-end encryption is a viable security countermeasure, but requires the cloud to compute on encrypted data.

TEEs such as SGX enclaves (cf. Section 3.4.1) promise to provide a secure environment in which data can be decrypted and then processed. However, software vulnerabilities give attackers ample opportunity to execute arbitrary code in the enclave program [Lee+17a]. These attacks can modify the control and data flow of the program and leak secrets from the enclave via untrusted memory or SGX side-channels (cf. Section 3.4.2). Since the number of software vulnerabilities grows with the size of the code base, it is advisable to keep the TCB as small as possible. Hence, it is not a good idea to outsource entire programs to an SGX enclave.

Consider the following dataflow modification attack that efficiently leaks a secret x in its entirety. Assume an encrypted variable $\mathsf{Enc}(x)$ in the domain $[0, N-1]$ is compared to $N/2-1$. The "then" branch is taken if it is lower or equal; the "else" branch otherwise. This can be observed, for example, by the branch shadowing attack presented by Lee et al. [Lee+17b]. The observation of this behavior leaks whether $x \leq N/2-1$. This becomes quite problematic when assuming a strong, active adversary that can modify the control and data flow. The adversary may then create constants $\mathsf{Enc}(\bar{x})$ for $\bar{x} \in \{N/4, N/8, N/16, \ldots, 1\}$ in the program code, add those to the variable $\mathsf{Enc}(x)$ and re-run the control-flow branch. This way, by consecutively adding or subtracting the constants, the adversary can conduct a binary search for the encrypted value.

As a defense for this attack of modifying the dataflow, we introduce the concept of dataflow authentication (DFAuth). We instrument each control-flow decision variable with a label (broadly speaking: a MAC tag), such that only variables with a pre-approved dataflow can be used in the decision. Variables carry unique identifiers that are preserved and checked during the encrypted operations. This prevents an adversary from deviating from the dataflow in ways that would allow attacks such as the one we described above. Note that a program may still have *intentional* leaks introduced by the programmer. However, DFAuth restricts the leakage of any program to these intended leaks by the programmer which the programmer could avoid, e.g., by using appropriate algorithms such as data-oblivious ones (cf. Section 3.2.2). In essence, the technique restricts the information flows to those that are equivalent to the original program's information flows.

FHE (cf. Section 3.1.4) would be an alternative to compute on encrypted data without the drawback of data leaks. However, due to its high computational complexity, researchers are seeking efficient alternatives that offer similar security. Fortunately, PHE can efficiently perform additively and multiplicatively homomorphic operations on encrypted data. Furthermore, if we reveal the control flow[1] of a program instead of computing a circuit, efficient computation seems feasible. Several proposals for program transformation into such encrypted computations have been made (cf. Section 3.3). MrCrypt [Tet+13], JCrypt [DMD16] and AutoCrypt [Top+13] each offer an increasing set of programs that can be computed on encrypted data. To support encrypted computation on all programs, one needs to convert between different homomorphic encryption schemes. These conversions are very small routines, such that one can scrutinize their code and implement them safely in a *trusted module* likely without any software vulnerabilities.

In this way, we combine the benefits of PHE with a small TCB and the efficiency of unprotected program execution. Our re-encryption routines are small and program-independent and are run protected in the trusted module whereas the program runs efficiently on homomorphic encrypted values in unprotected memory. We take care not to destroy the benefits of outsourcing. The verification of labels is constant time and does not depend on the homomorphic

[1]Note that any control-flow decision on an encrypted variable is an intentional leak by the programmer.

Approach	Support for Control Flow	Low Computational Overhead	Program-Independent Trusted Code Base
FHE	○	○	●
SGX only	●	●	○
AutoCrypt	◐	●	●
DFAuth	●	●	●

Table 5.1: Comparison of DFAuth to the most relevant alternative approaches computing on encrypted data.

computation. To this end, we introduce our own homomorphic authenticated symmetric encryption (HASE) scheme and the complementary concept of trusted authenticated ciphertext operations (TACO).

For a summary of key properties provided by DFAuth and a comparison to the most relevant alternative approaches for computation on encrypted data, refer to Table 5.1. Note that FHE does not require any trusted code to be executed by the untrusted evaluator (i.e., the CSP in our use case). Also note that AutoCrypt only supports control-flow decisions on encrypted *input* variables. DFAuth extends the state of the art to those programs performing control-flow decisions based on encrypted intermediate variables.

5.2 Definitions

In order to understand the security of DFAuth, we first introduce the overall adversary model considered, the algorithms that HASE offers and the security it guarantees.

5.2.1 Adversary Model

We consider a scenario between a trusted client and an untrusted cloud server, which has a trusted (hardware) module, e.g., an SGX enclave. Figure 5.1 depicts the process and its trust boundaries. The client wishes to execute a program at the cloud server with sensitive input data. Our security objective is to leak only the information about the inputs to the cloud server that can be inferred from the program's executed control flow.

We distinguish two phases of this outsourced computation: setup and runtime. First, the client chooses the keys for the encryption of its inputs in our HASE scheme (A). Then the client transforms the intended program using a specialized DFAuth-enabled compiler (B) and uploads it to the cloud. The server deploys some parts of the program into the trusted module which the client verifies by remote attestation (C). This concludes the setup phase.

In the runtime phase, the client can execute – multiple times if it wishes – the program on inputs of its choice. It encrypts the inputs using the information from the compiled program and sends the ciphertexts to the cloud server (1–2). The cloud server now executes the program (3). We assume an active adversary

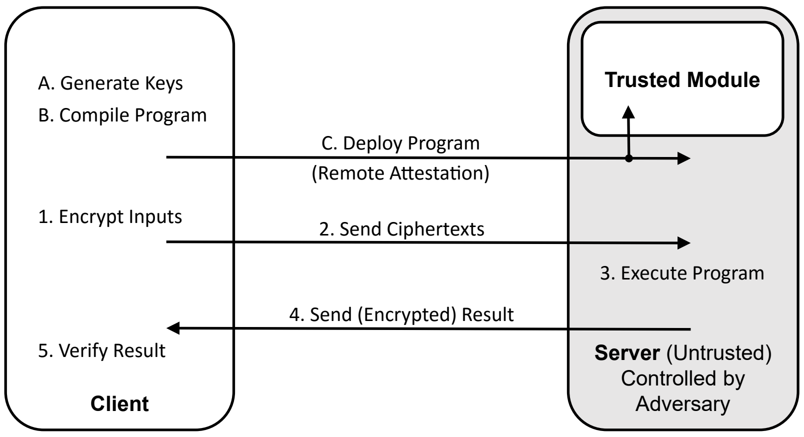


Figure 5.1: Dataflow Authentication System Overview

controlling the server who can

- *read* the contents of all variables and the program text, except in the trusted module.
- *modify* the contents of all variables and the program, except in the trusted module.
- *continuously observe and modify* the control flow, e.g., by breaking the program, executing instructions step-by-step and modifying the instruction pointer, except in the trusted module.
- do all of this arbitrarily *interleaved.*

After the execution of the program the server returns an encrypted result to the client (4). The client can then verify the result of the computation (5).

We ensure the following security property: The server has *learnt nothing beyond the intended information flow* of the program to unclassified memory locations (*interference equivalence* as presented in Section 5.3).

5.2.2 Homomorphic Authenticated Symmetric Encryption

In this section, we define the syntax, correctness and security of a homomorphic authenticated symmetric encryption (HASE) scheme. For security, we separately define confidentiality and authenticity.

Syntax and Correctness The syntax of a HASE scheme is defined as an adaptation of asymmetric homomorphic encryption (Definition 35). HASE, however, does not make use of the public-key property of asymmetric encryption, but considers a symmetric encryption setting in which the same secret key is used for encryption and decryption (cf. Section 2.2.2). Hence, the Gen algorithm does not output a public encryption key. Partially homomorphic encryption capabilities

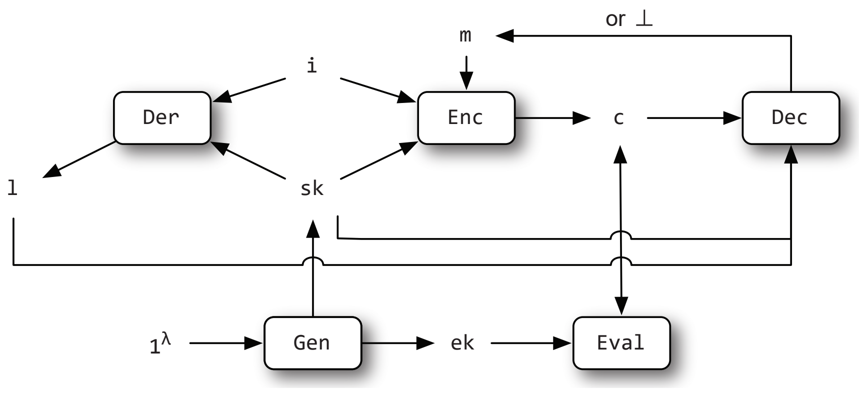


Figure 5.2: HASE Syntax Overview

are captured by an explicit Eval algorithm, which takes into account additional public parameters in the form of an evaluation key. The Gen algorithm hence also outputs an evaluation key. HASE implements message authentication (cf. Section 2.2.3) as follows: The encryption algorithm Enc takes an identifier parameter in addition to a message. Pre-approval of dataflows is implemented using an additional label derivation algorithm Der. The Der algorithm takes a set of identifiers as input and outputs a label. The decryption algorithm Dec takes a label parameter in addition to a ciphertext and uses the label to determine whether homomorphic evaluation followed the pre-approved dataflow.

We now formally define the syntax of a HASE scheme. An overview of all operations involved is provided in Figure 5.2.

Definition 36 (HASE Syntax)**.** *Let $\mathcal{I}$ be the set of identifiers. A homomorphic authenticated symmetric encryption (HASE) scheme is a tuple of PPT algorithms* (Gen, Enc, Eval, Der, Dec) *such that:*

- *The* key-generation algorithm Gen *takes the security parameter 1^λ as input and outputs a key pair (ek, sk) consisting of a* public evaluation key *ek and a* secret key *sk. The evaluation key implicitly defines a commutative plaintext group $(\mathcal{M}, \oplus)$, a commutative ciphertext group $(\mathcal{C}, \otimes)$ and a commutative label group $(\mathcal{L}, \diamond)$.*

- *The* encryption algorithm Enc *takes a secret key sk, a plaintext message $m \in \mathcal{M}$ and an identifier $i \in \mathcal{I}$ as input and outputs a ciphertext $c \in \mathcal{C}$.*

- *The* evaluation algorithm Eval *takes an evaluation key ek and a set of ciphertexts $C \subseteq \mathcal{C}$ as input and outputs a ciphertext $\hat{c} \in \mathcal{C}$.*

- *The deterministic* label derivation algorithm Der *takes a secret key sk and a set of identifiers $I \subseteq \mathcal{I}$ as input and outputs a secret label $l \in \mathcal{L}$.*

- *The deterministic* decryption algorithm Dec *takes a secret key sk, a ciphertext $c \in \mathcal{C}$ and a secret label $l \in \mathcal{L}$ as input and outputs a plaintext message $m \in \mathcal{M}$ or $\perp$ on decryption error.*

HASE correctness requires – except with negligible probability – the decryption of a ciphertext resulting from a legitimate homomorphic evaluation, i.e., an evaluation with a pre-approved dataflow, to result in the same plaintext as if all operations were performed directly on the corresponding plaintexts.

Definition 37 (HASE Correctness). *Let* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Der}, \mathsf{Dec})$ *be a HASE scheme,* $\mathcal{M}$ *the plaintext group and* $\mathcal{I}$ *the set of identifiers. We say that* Π *is* correct *if for all* $(m_1, \ldots, m_n) \in \mathcal{M}^n$ *with associated unique identifiers* $(i_1, \ldots, i_n) \in \mathcal{I}^n$ *there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\Pr\left[\hat{m} = \bigoplus_{j \in [1,n]} m_j \;\middle|\; \begin{array}{l} (ek, sk) \leftarrow \mathsf{Gen}(1^\lambda) \\ \forall j \in [1,n] : c_j \leftarrow \mathsf{Enc}(sk, m_j, i_j) \\ l := \mathsf{Der}(sk, \{i_1, \ldots, i_n\}) \\ \hat{c} \leftarrow \mathsf{Eval}(ek, \{c_1, \ldots, c_n\}) \\ \hat{m} := \mathsf{Dec}(sk, \hat{c}, l) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

where the probability is taken over the randomness of all algorithms.

Security Definitions We define confidentiality in terms of indistinguishability and authenticity in terms of unforgeability. Indistinguishability of HASE schemes is defined as an adaptation of the IND-CPA security definition for symmetric encryption schemes (Definition 5). Unforgeability of HASE schemes is based on the *unforgeable encryption* definition (Definition 15).

Definition 38 (HASE-IND-CPA). *A HASE scheme* Π *has* indistinguishable encryptions under a chosen-plaintext attack*, or is* CPA-secure*, if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{A},\Pi}(\lambda) := \left|\Pr\left[\text{ExpHASE}^{\text{IND-CPA}}_{\mathcal{A},\Pi}(\lambda) = 1\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda)$$

The experiment is defined as follows:

```
$\text{ExpHASE}^{\text{IND-CPA}}_{\mathcal{A},\Pi}(\lambda)$
$(ek, sk) \leftarrow \Pi.\mathsf{Gen}(1^\lambda)$
$(m_0, m_1, i_0, i_1, \mathsf{st}) \leftarrow \mathcal{A}^{\Pi.\mathsf{Enc}(sk,\cdot,\cdot)}(1^\lambda, ek)$
$b \leftarrow_\$ \{0,1\}$
$c \leftarrow \Pi.\mathsf{Enc}(sk, m_b, i_b)$
$b' \leftarrow \mathcal{A}^{\Pi.\mathsf{Enc}(sk,\cdot,\cdot)}(1^\lambda, c, \mathsf{st})$
**return** $b = b'$
```

Definition 39 (HASE-UF-CPA). *A HASE scheme* Π *is* unforgeable under a chosen-plaintext attack*, or just* unforgeable*, if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\mathsf{Adv}^{\text{UF-CPA}}_{\mathcal{A},\Pi}(\lambda) := \Pr\left[\text{ExpHASE}^{\text{UF-CPA}}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

The experiment is defined as follows:

```
$\text{ExpHASE}^{\text{UF-CPA}}_{\mathcal{A},\Pi}(\lambda)$
$S := \{\}$
$(ek, sk) \leftarrow \Pi.\mathsf{Gen}(1^\lambda)$
$(c, I) \leftarrow \mathcal{A}^{E(sk,\cdot,\cdot)}(1^\lambda, ek)$
$l := \Pi.\mathsf{Der}(sk, I)$
$m := \Pi.\mathsf{Dec}(sk, c, l)$
$\tilde{m} := \bigoplus_{(m',i)\in S, i\in I} m'$
return $m \neq \perp \wedge m \neq \tilde{m}$
```

```
$E(sk, m, i)$
if $i \in \pi_2(S)$ then
    return $\perp$
else
    $S := S \cup \{(m, i)\}$
    $c \leftarrow \Pi.\mathsf{Enc}(sk, m, i)$
    return $c$
```

The adversary returns a ciphertext c and a set of identifiers I. The adversary is successful if and only if two conditions are met. First, c has to successfully decrypt under the label derived from I. Second, the resulting plaintext m must be different from the plaintext $\tilde{m}$ resulting from the application of the plaintext operation to the set of plaintexts corresponding to I. Note that by controlling I the adversary controls which elements of S are used for the evaluation resulting in $\tilde{m}$.

5.3 Dataflow Authentication

We introduce dataflow authentication (DFAuth) using an example. Consider the following excerpt from a Java program:

```
a = b + c;
d = a * e;
if (d > 42)
    f = 1;
else
    f = 0;
```

First DFAuth performs a conversion to single static assignment (SSA) form [AWZ88]: assign each variable at exactly one code location; create atomic expressions; introduce fresh variables if required. In the example, DFAuth changes the program to the following:

```
a = b + c;
d = a * e;
d1 = d > 42;
if (d1)
    f1 = 1;
else
    f2 = 0;
f = phi(f1,f2);
```

As usual in SSA, phi is a specially interpreted merge function that combines the values of both assignments to f, here denoted by f1 and f2.

DFAuth then performs a type inference similar to JCrypt [DMD16] and AutoCrypt [Top+13]. As a result of this inference, each variable and constant is assigned an encryption type of $\{add, mul, cmp\}$. At runtime, each constant and variable value will be encrypted according to the appropriate type. HASE implements multiplicative homomorphic encryption *mul* and its operations directly, while it implements additive homomorphic encryption *add* using exponentiation. Comparisons *cmp* are implemented in the trusted module. Our experiments show that this is more efficient than performing the comparison in the program space using conversion to searchable or functional encryption. An attacker observing user space will hence only see encrypted variables and constants, but can observe the control flow. Actual data values are hidden from the attacker.

Combinations of multiple operations, however, require additional work. Every time a variable is encrypted in one encryption type (e.g., additive), but is later used in a different one (e.g., multiplicative), DFAuth must insert a conversion. The resulting program in our running example looks as follows:

Listing 5.1: Example as executed on the server

```
a = b + c;
a1 = convertToMul(a, "a1");
d = a1 * e;
d1 = convertToCmpGT42(d, "d1");
if (d1)
    f1 = 1;
else
    f2 = 0;
f = phi(f1, f2);
```

The first conversion is necessary because the variable `a` must be converted from additive to multiplicative homomorphic encryption. The resulting re-encrypted value is stored in `a1`. For security reasons, the decryption performed by the conversion routine must be sensitive to the variable identifier it is assigned to. A unique label must be introduced to make the decryption routine aware of the code location. DFAuth can use the left-hand-side variable's identifier (`"a1"` in this example), because it introduced unique names during SSA conversion. Using this variable identifier, the conversion routine can retrieve the corresponding label of the HASE encryption stored in the memory protected by the trusted module.

Any branch condition is also treated as a conversion that leaks the result of the condition check. In the example, DFAuth introduces the variable `d1` to reflect this result:

```
d1 = convertToCmpGT42(d, "d1");
```

To simplify the exposition, we assume that our compiler inlines this comparison into a special routine `convertToCmpGT42`. In the general case, a binary comparison on two variables `x` and `y` would result in a call to a routine `convertToCmp(x, y, "z")`. We show the full algorithm in Listing 5.4 in Section 5.6 which is generic for all comparisons and in case of comparison to a constant looks up this constant in an internal table protected by the trusted module. We need to protect constants in comparisons, since if they were part of the program text, they could be modified by the adversary.

As mentioned before, the security challenge of such conversions to *cmp* is that they leak information about the encrypted variables, and particularly that active adversaries who can modify the control and data flow can exploit those leaks to restore the variables' plaintext. In this work, we thus propose to restrict the dataflow using DFAuth. This allows such conversions in a secure way by enforcing that encrypted variables can be decrypted only along the program's original dataflow. The approach comprises two steps. First, happening at compile time, for each conversion DFAuth pre-computes the Der algorithm (cf. Definition 36) on the operations in the code. In the conversion `convertToMul(a, "a1")` (at line 2 in our example), DFAuth computes the label

$$l2 = \mathsf{Der}(sk, \{\texttt{"b"}, \texttt{"c"}\})$$

and in the conversion at line 4

$$l4 = \mathsf{Der}(sk, \{\texttt{"a1"}, \texttt{"e"}\})$$

Here the second argument to Der is the multi-set of variable identifiers involved in the unique computation preceding the conversion. We use a multi-set and not a vector, because all our encrypted operations are commutative. The compiler computes labels for all variables and constants in the program.

At runtime the computed labels as well as the secret key *sk* must be kept secret from the attacker, which is why both are securely transferred to, and stored in, the trusted module during the setup phase. The trusted module registers the secret labels under the respective identifier, for example, associating label $l4$ with identifier `"d1"`.

All conversion routines run within the trusted module. They retrieve a secret label for an identifier with the help of a `labelLookup(id)` function. In particular, when the program runs and a conversion routine is invoked, the trusted module looks up and uses the required labels for decryption. In the example at line 4, the call to `convertToCmpGT42` internally invokes the decryption operation `Dec(sk, d, l4)` using secret label `l4` retrieved for variable identifier `"d1"`:

```
1  convertToCmpGT42(d, "d1") {
2      l4 = labelLookup("d1");
3      x = Dec(sk, d, l4);
4      if (x == fail)
5          stop;
6      return (x > 42);
7  }
```

Note that in this scheme, the trusted module returns the result of the comparison in the clear. In this case, however, leaking the branch decision is *secure*, as HASE guarantees that any active attack that would yield the adversary a significant advantage will be reliably detected.

Let us assume an attacker that attempts to modify the program's data or control flow to leak information about the encrypted plaintexts, for instance, using a binary search as described in Section 5.1. The attacker is not restricted to the compiled instructions in the program, and can also try to "guess" the result of cryptographic operations as the adversary in experiment $\mathrm{ExpHASE}_{\mathcal{A},\Pi}^{\text{IND-CPA}}$. This modification to binary search can only succeed if the decryption operations

Listing 5.2: Example modified by the attacker

```
a = b + c;
a1 = convertToMul(a, "a1");
g1 = a1 * e;                 // changed
for (i = n..1) {             // inserted
    g = phi(g1, g3);         // inserted
    d = g + 2^i;             // inserted
    d1 = convertToCmpGT42(d, "d1");
    if (d1) {
        f1 = 1;
        g2 = g3 - 2^i;       // inserted
    } else {
        f2 = 0;
        g3 = phi(g, g2);     // inserted
        f = phi(f1,f2);
        leak(f);             // inserted
    }
}
```

Dec in `convertToCmpGT42` (or other conversion routines) succeed. The adversary can minimize the Dec operations, e.g., by not introducing new calls to conversion routines, but given the scheme defined above, any attempt to alter the dataflow on encrypted variables will cause Dec to fail: Assume that an attacker inserts code in Listing 5.1 to search for the secret value `d` resulting in the code shown in Listing 5.2. We only use this code to illustrate potential attacks and ignore the fact that the attacker would need access to the encrypted constants (`2^i`) and needs to guess the result of the homomorphic addition operation on the ciphertexts. However, given these capabilities, the attacker could try to observe the control flow – simulated by our statement `leak(f)` – which then would in turn leak the value of `d`.

This code will only execute if each variable decryption succeeds, but decryption for instance of `d1` will succeed only if it was encrypted with the same label `l4` that was associated with `d1` at load time. Since the trusted module keeps the labels and the key *sk* secret, the attacker cannot possibly forge the required label at runtime. Moreover, in the attacker-modified program, the encryption must fail due to the altered data dependencies: in the example, the input `d` to `convertToCmpGT42` has now been derived from `g3` and `i` instead of `a1` and `e`, which leads to a non-matching label for `d`. In result, the decryption in the conversion routine `convertToCmpGt42` will fail and stop program execution before any unintended leakage can occur.

General Security Argument The way in which we derive labels from dataflow relationships enforces a notion of *interference equivalence.* A program P is said to be *non-interferent* [Smi07], if applied to two different memory configurations M_1, M_2 that are equal w.r.t. their low, i.e. unclassified (unencrypted), memory locations, $M_1 =_L M_2$ for short, then also the resulting memory locations after program execution must show such low-equivalency: $P(M_1) =_L P(M_2)$. Non-interference holds if and only if there is no information flow from high, i.e. classified (encrypted), values to low memory locations. While this is a semantic property, previous research has shown that one can decide non-interference also through a structural analysis of programs, through so-called program dependency graphs (PDGs) that capture the program's control and data flow [WLS09]. In this view, a program is *non-interferent* if the PDG is free of paths from high to low memory locations.

In the setting considered in this work one must assume that the executed program *before encryption* already shows interference for some memory locations, e.g., because the program is, in fact, intended to declassify some limited information (notably control-flow information). Let $M \downarrow C$ denote a projection of memory configuration M onto all (classified) memory locations C that are not declassified that way. Then even in the setting here it holds for any program P and any memory configurations M_1, M_2 that $P(M_1 \downarrow C) =_L P(M_2 \downarrow C)$.

The main point of the construction proposed in this work is that any program that an attacker can produce, and that would lead to the same computation of labels (and hence decryptable data) as the original program, cannot produce any more information flows than the original program. Let us denote by tr a program transformation conducted by the attacker, e.g., the transformation explained above, which inserted a binary search. Then the property we would like to obtain is that:

$$\forall M_1, M_2, tr : P(M_1 \downarrow C) =_L P(M_2 \downarrow C) \\ \implies (tr(P))(M_1 \downarrow C) =_L (tr(P))(M_2 \downarrow C)$$

In other words: disregarding the explicitly declassified information within C, the transformed program does not leak any additional information, i.e., the adversary cannot learn any additional information about the encrypted data. Let us assume for a moment that the above equation did not hold. If that were true then there would exist a transformation tr that would cause the transformed program $tr(P)$ to compute values in at least one low memory location despite low-equivalent inputs. But this is impossible, as any such transformation would necessarily have to insert additional PDG-edges, destroying at least one label computation, and hence invalidating our HASE-UF-CPA security proof.

Result Verification Note that the client can verify the result of the computation using a simple check on the variable's label – just as the conversion routine does. The result is just another variable, which albeit not being converted, can be checked for correct dataflow computation. That way, a client can ensure that it receives a valid output of the program.

5.4 HASE Constructions

In this section, we provide two constructions of HASE schemes. Section 5.4.1 introduces *Multiplicative HASE* allowing homomorphic multiplication on integers. Section 5.4.2 presents *Additive HASE* supporting homomorphic addition on integers. Finally, Section 5.4.3 shows the security of our schemes under their assumptions.

5.4.1 Multiplicative HASE

Our first construction is based on the Elgamal public-key encryption scheme (Construction 1). We do not make use of the public-key property of the scheme, but extend ciphertexts with a third group element working as a homomorphic authenticator (i.e., a MAC tag).

Construction 2 (Multiplicative HASE)**.** *Let $\mathcal{G}$ be a group generation algorithm (Definition 7). Define a HASE scheme using the following PPT algorithms:*

- Gen*: on input 1^λ obtain $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^\lambda)$. For a pseudorandom function family $H : \mathcal{K} \times \mathcal{I} \to \mathbb{G}$ choose $k \leftarrow\$\ \mathcal{K}$. Choose $a, x, y \leftarrow\$\ \mathbb{Z}_q$ and compute $h := g^x$, $j := g^y$. The evaluation key is $\mathbb{G}$, the secret key is $(\mathbb{G}, q, g, a, x, y, h, j, k)$. The plaintext group is $(\mathcal{M}, \oplus) := (\mathbb{G}, \cdot)$ where $\cdot$ is the group operation in $\mathbb{G}$. The ciphertext group is $(\mathbb{G}^3, \otimes)$ where we define $\otimes$ to denote the component-wise application of $\cdot$ in $\mathbb{G}$. The label space is $(\mathbb{G}, \cdot)$.*

- Enc*: on input a secret key $sk = (\mathbb{G}, q, g, a, x, y, h, j, k)$, a message $m \in \mathbb{G}$ and an identifier $i \in \mathcal{I}$. Choose $r \leftarrow\$\ \mathbb{Z}_q$ and obtain the label $l = H(k, i)$. Compute $u := g^r$, $v := h^r \cdot m$ and $w := j^r \cdot m^a \cdot l$. Output the ciphertext (u, v, w).*

- Eval*: on input an evaluation key $\mathbb{G}$ and a set of ciphertexts $C \subseteq \mathcal{C}$ compute the ciphertext*
$$c := \bigotimes_{c' \in C} c'$$
and output c.

- Der*: on input a secret key $(\mathbb{G}, q, g, a, x, y, h, j, k)$ and a set of identifiers $I \subseteq \mathcal{I}$ compute the label*
$$l := \prod_{i \in I} H(k, i)$$
and output l. Note that here Π denotes the repeated application of the group operation $\cdot$ in $\mathbb{G}$.

- Dec*: on input a secret key $(\mathbb{G}, q, g, a, x, y, h, j, k)$, a ciphertext $c = (u, v, w)$ and a secret label $l \in \mathbb{G}$. First compute $m := u^{-x} \cdot v$, then $t := u^y \cdot m^a \cdot l$. If t equals w output m, otherwise output $\perp$.*

It is well known that the Elgamal encryption scheme is homomorphic with regard to the group operation in $\mathbb{G}$. As can be easily seen, this property is inherited by our construction. For the original Elgamal scheme, $\mathbb{G}$ is most commonly instantiated either as $\mathbb{G}_q$, the q-order subgroup of quadratic residues of $\mathbb{Z}_p^*$ for some prime $p = 2q + 1$ (with q also prime), or as an elliptic curve over some

q-order finite field. In the latter case, the group operation is elliptic curve point addition and the ability to perform point addition in a homomorphism serves no useful purpose in our context. Instantiating $\mathbb{G}$ as $\mathbb{G}_q$ on the other hand enables homomorphic multiplication on the integers.

5.4.2 Additive HASE

Our second construction supports homomorphic integer addition and is obtained by applying a technique proposed by Hu et al. [HMS12] to Multiplicative HASE (Construction 2). The basic idea is to consider plaintexts to be element of $\mathbb{Z}_q$ instead of $\mathbb{G}$ and to encrypt a given plaintext m by first raising the generator g to the power of m and then encrypting the resulting group element in the usual way. In detail, this means computing ciphertexts of the form $(g^r, h^r g^m)$ rather than $(g^r, h^r m)$. To see that the resulting scheme is homomorphic with regard to addition on $\mathbb{Z}_q$, consider what happens when the group operation is applied component-wise to two ciphertexts:

$$(g^{r_1} \cdot g^{r_2}, h^{r_1} g^{m_1} \cdot h^{r_2} g^{m_2}) = (g^{r_1+r_2}, h^{r_1+r_2} \cdot g^{m_1+m_2})$$

Unfortunately, decryption now involves computing discrete logarithms with respect to base g, which must be difficult for sufficiently large exponents in order for the DDH problem (cf. Definition 8) to be hard relative to $\mathcal{G}$. Hu et al. [HMS12] keep exponents small enough for discrete logarithm algorithms to terminate within reasonable time despite their exponential asymptotic running time. They do so by unambiguously decomposing plaintexts m into t smaller plaintexts m_e ($e \in [1, t]$) via means of the Chinese remainder theorem (CRT) and then encrypting each m_e separately. Although doing so increases the ciphertext size roughly by a factor of t in comparison to Construction 2, this drawback can be compensated by instantiating $\mathbb{G}$ as an elliptic curve group since the homomorphic operation is on $\mathbb{Z}_q$ rather than $\mathbb{G}$. At a comparable security level, group elements of elliptic curves can be represented using a fraction of bits [Sma+14].

We now provide the full details of our *Additive HASE* construction. Note that the authenticator only requires constant (i.e., independent of t) ciphertext space and can be verified without discrete logarithm computation. Although we consider instantiating $\mathbb{G}$ as an elliptic curve group, we keep writing the group operation multiplicatively.

Construction 3 (Additive HASE)**.** *Let $\mathcal{G}$ be a group generation algorithm (Definition 7). Define a HASE scheme using the following PPT algorithms and the* Eval *algorithm from Construction 2:*

- Gen*: on input 1^λ obtain $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^\lambda)$. For a pseudorandom function family $H : \mathcal{K} \times \mathcal{I} \to \mathbb{Z}_q$ choose $k \leftarrow_\$ \mathcal{K}$. Choose $\{d_1, \dots, d_t\} \subset \mathbb{Z}^+$ such that $d := \prod_{e=1}^{t} d_e < q$ and $\forall e \neq j : \gcd(d_e, d_j) = 1$. Define $D := (d_1, \dots, d_t, d)$. Choose $a, x, y \leftarrow_\$ \mathbb{Z}_q$ and compute $h := g^x$, $j := g^y$. The evaluation key is $\mathbb{G}$, the secret key is $(\mathbb{G}, q, g, a, x, y, h, j, k, D)$. The plaintext group is $(\mathcal{M}, \oplus) := (\mathbb{Z}_d, +)$. The ciphertext group is $(\mathbb{G}^{2(t+1)}, \otimes)$ where $\otimes$ denotes the component-wise application of $\cdot$ in $\mathbb{G}$. The label space is $(\mathbb{G}, \cdot)$.*

- Enc*: on input a secret key* $sk = (\mathbb{G}, q, g, a, x, y, h, j, k, D)$*, a message* $m \in \mathbb{Z}_d$ *and an identifier* $i \in \mathcal{I}$*. Obtain the label* $l := H(k, i)$*. For* $e := 1, \ldots, t$*:*
  - *Compute* $m_e := m \bmod d_e$*.*
  - *Choose* $r_e \leftarrow\!\!\$\, \mathbb{Z}_q$
  - *Compute* $u_e := g^{r_e}$
  - *Compute* $v_e := h^{r_e} \cdot g^{m_e}$

  Choose $r \leftarrow\!\!\$\, \mathbb{Z}_q$*. Compute* $s := g^r$ *and* $w := j^r \cdot g^{m \cdot a} \cdot l$*. Output the ciphertext* $(u_1, v_1, \ldots, u_t, v_t, s, w)$*.*

- Der*: on input a secret key* $(\mathbb{G}, q, g, a, x, y, h, j, k, D)$ *and a set of identifiers* $I \subseteq \mathcal{I}$ *compute the label*

  $$l := \prod_{i \in I} g^{H(k,i)}$$

  and output l*.*

- Dec*: on input a secret key* $(\mathbb{G}, q, g, a, x, y, h, j, k, D)$*, a ciphertext* $(u_1, v_1, \ldots, u_t, v_t, s, w)$ *and a secret label* $l \in \mathbb{G}$*. Parse* $D = (d_1, \ldots, d_t, d)$*. First compute* $m_e := \log_g(v_e {u_e}^{-x})$ *for* $e = 1, \ldots, t$*, then recover*

  $$m := \sum_{e=1}^{t} m_e \frac{d}{d_e} \left(\frac{d}{d_e}^{-1} \bmod d_e \right) \bmod d.$$

  If $s^y \cdot g^{m \cdot a} \cdot l = w$ *then output* m*, else output* $\perp$*. Note that* $\log_g$ *denotes the discrete logarithm with respect to base* g*.*

5.4.3 Security Reductions

Theorem 2 (Multiplicative HASE-IND-CPA)**.** *Let* Π *be Construction 2. If the DDH problem is hard relative to* $\mathcal{G}$ *and* H *is a PRF as described in* $\Pi.\mathsf{Gen}$*, then* Π *is CPA-secure.*

Proof. Let $\Pi, \mathcal{G}, H$ be as described and let $\mathcal{A}$ be a PPT adversary. We use a sequence of games to show that $\mathcal{A}$'s advantage $\mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{A},\Pi}(\lambda)$ is negligible in λ. For Game n we use S_n to denote the event that $b = b'$. The final game and the encryption oracle used in all games are given in Figure 5.3.

Game 0. This is the original experiment from Definition 38 except that instead of relying on Π the challenger performs the exact same computations on its own. Clearly, $\mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{A},\Pi}(\lambda) = |\Pr[S_0] - \frac{1}{2}|$.

Game 1 (Indistinguishability-Based Transition). Instead of deriving the label used in the third component of the challenge ciphertext using the pseudorandom function $H : \mathcal{K} \times \mathcal{I} \to \mathbb{G}$ for some random $k \leftarrow\!\!\$\, \mathcal{K}$, we make use of a random function $f \leftarrow\!\!\$\, \mathcal{F}$ from the set of functions $\mathcal{F} = \{F : \mathcal{I} \to \mathbb{G}\}$.

```
$\text{Game3}_{\mathcal{A}}^{\text{IND-CPA}}(\lambda)$
$(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^\lambda)$
$a, x, y \leftarrow_\$ \mathbb{Z}_q, k \leftarrow_\$ \mathcal{K}$
$h := g^x, j := g^y$
$ek := \mathbb{G}$
$sk := (\mathbb{G}, q, g, a, x, y, h, j, k)$
$(m_0, m_1, i_0, i_1, \mathsf{st}) \leftarrow \mathcal{A}^{E(sk,\cdot,\cdot)}(1^\lambda, ek)$
$b \leftarrow_\$ \{0,1\}$
$r, \boxed{s}, \boxed{z} \leftarrow_\$ \mathbb{Z}_q$
$l := \boxed{g^s}$
$c := (g^r, \boxed{g^z} \cdot m_b, j^r \cdot {m_b}^a \cdot l)$
$b' \leftarrow \mathcal{A}^{E(sk,\cdot,\cdot)}(1^\lambda, c, \mathsf{st})$
**return** $b = b'$
```

```
$E(sk, m, i)$
**parse** $sk = (\mathbb{G}, q, g, a, x, y, h, j, k)$
$r \leftarrow_\$ \mathbb{Z}_q$
$l := H(k, i)$
$c := (g^r, h^r \cdot m, j^r \cdot m^a \cdot l)$
**return** $c$
```

Figure 5.3: Final security experiment used in Multiplicative HASE-IND-CPA proof. Changes compared to the first experiment are highlighted.

We construct a polynomial time algorithm $\mathcal{B}$ distinguishing between a PRF (for a random key) and a random function using $\mathcal{A}$ as a black box. If $\mathcal{B}$'s oracle is a pseudorandom function, then the view of $\mathcal{A}$ is distributed as in Game 0 and we have $\Pr[S_0] = \Pr\left[\mathcal{B}^{\mathcal{A},H(k,\cdot)}(1^\lambda) = 1\right]$ for some $k \leftarrow_\$ \mathcal{K}$. If $\mathcal{B}$'s oracle is a random function, then the view of $\mathcal{A}$ is distributed as in Game 1 and thus we have $\Pr[S_1] = \Pr\left[\mathcal{B}^{\mathcal{A},f(\cdot)}(1^\lambda) = 1\right]$ for some $f \leftarrow_\$ \mathcal{F}$. Under the assumption that H is a PRF, $|\Pr[S_0] - \Pr[S_1]|$ is negligible.

Game 2 (Conceptual Transition). Because f is only evaluated on a single input i_b and f is a random function, the result is a random element of $\mathcal{G}$. Thus, instead of computing $l := f(i_b)$, we can compute $l := g^s$ for a random exponent $s \leftarrow_\$ \mathbb{Z}_q$. Since this is only a conceptual change, we have $\Pr[S_1] = \Pr[S_2]$.

Game 3 (Indistinguishability-Based Transition). In the challenge ciphertext we replace $h^r = g^{xr}$ with a random group element g^z generated by raising g to the power of a random $z \leftarrow_\$ \mathbb{Z}_q$.

We construct a polynomial time distinguishing algorithm $\mathcal{D}$ solving the DDH problem that interpolates between Game 2 and Game 3. If $\mathcal{D}$ receives a real triple $(g^\alpha, g^\beta, g^{\alpha\beta})$ for $\alpha, \beta \leftarrow_\$ \mathbb{Z}_q$, then $\mathcal{A}$ operates on a challenge ciphertext constructed as in Game 2 and thus we have

$$\Pr[S_2] = \Pr\left[\mathcal{D}^{\mathcal{A}}(\mathbb{G}, q, g, g^\alpha, g^\beta, g^{\alpha\beta}) = 1\right].$$

If $\mathcal{D}$ receives a random triple $(g^\alpha, g^\beta, g^\gamma)$ for $\alpha, \beta, \gamma \leftarrow_\$ \mathbb{Z}_q$, then $\mathcal{A}$ operates on a

```
$\text{Game2}^{\text{UF-CPA}}_{\mathcal{A},\mathcal{G},H}(\lambda)$
$S := \{\}$
$(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^\lambda)$
$a, x, y \leftarrow_{\$} \mathbb{Z}_q, k \leftarrow_{\$} \mathcal{K}, \boxed{f \leftarrow_{\$} \mathcal{F}}$
$h := g^x, j := g^y$
$ek := \mathbb{G}$
$sk := (\mathbb{G}, q, g, a, x, y, h, j, \boxed{f})$
$(c, I) \leftarrow \mathcal{A}^{E(sk,\cdot,\cdot)}(1^\lambda, ek)$
$l := \prod_{i \in I} \boxed{f(i)}$
parse $c = (u, v, w)$
$m := u^{-x} \cdot v$
$t := u^y \cdot m^a \cdot l$
$\tilde{m} := \bigoplus_{(m',i) \in S, i \in I} m'$
return $\boxed{t = w} \wedge m \neq \tilde{m}$
```

```
$E(sk, m, i)$
parse $sk = (\mathbb{G}, q, g, a, x, y, h, j, \boxed{f})$
if $i \in \pi_2(S)$ then
    return $\perp$
else
    $S := S \cup \{(m, i)\}$
    $r \leftarrow_{\$} \mathbb{Z}_q, l := \boxed{f(i)}$
    $c := (g^r, h^r \cdot m, j^r \cdot m^a \cdot l)$
    return $c$
```

Figure 5.4: Final security experiment used in Multiplicative HASE-UF-CPA proof. Changes compared to the first experiment are highlighted.

challenge ciphertext constructed as in Game 3 and thus we have

$$\Pr[S_3] = \Pr\left[\mathcal{D}^{\mathcal{A}}(\mathbb{G}, q, g, g^\alpha, g^\beta, g^\gamma) = 1\right].$$

In both cases $\mathcal{D}$ receives $(\mathbb{G}, q, g)$ output by $\mathcal{G}(1^\lambda)$. Under the assumption that the DDH problem is hard relative to $\mathcal{G}$, $|\Pr[S_2] - \Pr[S_3]|$ is negligible.

Conclusion. In the last game, the first component of the challenge ciphertext is trivially independent of the challenge plaintext as well as the challenge identifier. In the second component, g^z acts like a one-time pad and completely hides m_b. Similarly, $l = g^s$ acts like a one-time pad in the third component. Because the challenge ciphertext does not contain any information about m_b or i_b, we conclude that $\Pr[S_3] = \frac{1}{2}$. Overall we have that $\mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{A},\Pi}(\lambda) = \mathsf{negl}(\lambda)$. $\square$

Theorem 3 (Multiplicative HASE-UF-CPA)**.** *Let* Π *be Construction 2. If* H *is a PRF as described in* $\Pi.\mathsf{Gen}$*, then* Π *is unforgeable.*

Proof. Let $\Pi, \mathcal{G}, H$ be as described and let $\mathcal{A}$ be a PPT adversary. We use a sequence of games to show that $\mathcal{A}$'s advantage $\mathsf{Adv}^{\text{UF-CPA}}_{\mathcal{A},\Pi}(\lambda)$ is negligible in λ. For Game n we use S_n to denote the event that the adversary wins the game. The final game is illustrated in Figure 5.4.

Game 0. This is the original experiment from Definition 39 except that instead of relying on Π the challenger performs the exact same computations on its own. Clearly, $\mathsf{Adv}^{\text{UF-CPA}}_{\mathcal{A},\Pi}(\lambda) = |\Pr[S_0]|$.

Game 1 (Conceptual Transition). We eliminate the conditional statement introduced by the Dec algorithm by comparing t and w in the return statement.

Game 2 (Indistinguishability-Based Transition). We replace the pseudorandom function $H(k, \cdot)$ with a function $f(\cdot)$ chosen at random. Under the assumption that H is a PRF, we have that $|\Pr[S_1] - \Pr[S_2]|$ is negligible as in the previous security reduction in Theorem 2.

Conclusion. We show that $\Pr[S_2] = \mathsf{negl}(\lambda)$. Let X be the event that $\forall i \in I : \exists (m, i) \in S$, i.e., all identifiers have been used in encryption oracle queries.

In case event X does not happen, the challenger evaluates function f on at least one new argument. By the definition of f, the result is a random value in the image of f. This random group element acts as a one-time pad and makes l look random. Subsequently, t is also random from the point of view of the adversary. To win the experiment, $\mathcal{A}$ has to fulfill $t = w$. Because t is random, $\mathcal{A}$ cannot guess the correct w with probability better than $\frac{1}{q}$. Thus, we have

$$\Pr[S_2 \wedge \neg X] = \frac{1}{q} \cdot \Pr[\neg X] \ . \tag{5.1}$$

Recall that q is the order of $\mathbb{G}$ (of which w is an element) and both are output by the group generation algorithm $\mathcal{G}(1^\lambda)$. Also note that $\neg X$ holds when $\mathcal{A}$ performs no encryption queries at all.

Now consider the case when event X happens and let (c, I) be the output of the adversary. The set of identifiers I determines a label l and an expected message $\tilde{m}$. Furthermore, let $\tilde{c} = (\tilde{u}, \tilde{v}, \tilde{w})$ be the ciphertext resulting from the application of $\Pi.\mathsf{Eval}$ to ciphertexts identified by I. As $\tilde{c}$ is an honestly derived encryption of $\tilde{m}$, the following must hold:

$$\begin{aligned} \tilde{m} &= \tilde{u}^{-x} \cdot \tilde{v} \\ \tilde{w} &= \tilde{u}^{y} \cdot \tilde{m}^{a} \cdot l \\ &= (\tilde{u}^{y-x} \cdot \tilde{v})^{a} \cdot l \end{aligned} \tag{5.2}$$

Similarly, in order for $c = (u, v, w)$ to be accepted as a forgery regarding I, it must hold that:

$$w = (u^{y-x} \cdot v)^{a} \cdot l \tag{5.3}$$

for some $m := u^{-x} \cdot v \neq \tilde{m}$. Because $m \neq \tilde{m}$ we know that $\tilde{u}^{y-x} \cdot \tilde{v} \neq u^{y-x} \cdot v$ and $\tilde{w} \neq w$.

Combining equations (5.2) and (5.3) yields

$$\begin{aligned}\frac{\tilde{w}}{w} &= \frac{(\tilde{u}^{y-x} \cdot \tilde{v})^a \cdot l}{(u^{y-x} \cdot v)^a \cdot l} \\ &= \left(\frac{\tilde{u}^{y-x} \cdot \tilde{v}}{u^{y-x} \cdot v}\right)^a \end{aligned} \tag{5.4}$$

In order for c to be a forgery with regard to I, equation (5.4) needs to be satisfied. But since a is a random element of $\mathbb{Z}_q$, the probability that $\mathcal{A}$ can satisfy (5.4) is only $\frac{1}{q}$. Hence,

$$\Pr[S_2 \wedge X] = \frac{1}{q} \cdot \Pr[X] \ . \tag{5.5}$$

Summing up (5.1) and (5.5), we have

$$\Pr[S_2] = \Pr[S_2 \wedge \neg X] + \Pr[S_2 \wedge X] = \frac{1}{q}$$

and overall we have that $\mathsf{Adv}^{\text{UF-CPA}}_{\mathcal{A},\Pi}(\lambda) = \mathsf{negl}(\lambda)$ □

Theorem 4 (Additive HASE-IND-CPA)**.** *Let* Π *be Construction 3. If the DDH problem is hard relative to* $\mathcal{G}$ *and* H *is a PRF as described in* $\Pi.\mathsf{Gen}$*, then* Π *is CPA-secure.*

Proof Sketch. The security follows from Theorem 2. □

Theorem 5 (Additive HASE-UF-CPA)**.** *Let* Π *be Construction 3. If* H *is a PRF as described in* $\Pi.\mathsf{Gen}$*, then* Π *is unforgeable.*

Proof Sketch. The security follows from Theorem 3. □

5.5 Trusted Authenticated Ciphertext Operations

In this section, we complement DFAuth and HASE with trusted authenticated ciphertext operations (TACO), an alternative concept for operating on ciphertexts.

TACO is similar to HASE, but evaluations in TACO are defined as a secret key operation, which offers some advantages. First, constructions do not have to rely on homomorphic encryption, but can make use of more efficient symmetric encryption schemes. Second, the TACO syntax is more powerful and allows to perform multiplication, addition and other operations such as division using the same generic construction. However, these properties are a trade-off: Since the evaluation algorithm depends on the secret key, it must be run by a trusted party, i.e., the client or the trusted module in our setting (cf. Section 5.2).

We define the syntax and correctness of a TACO scheme in Section 5.5.1. Section 5.5.2 provides our TACO security definitions. Section 5.5.3 presents a construction of a TACO scheme and Section 5.5.4 shows its security properties.

5.5.1 Syntax and Correctness

The syntax is similar to that of HASE. An important difference is that evaluation can be performed for multiple operations which do not necessarily have to be commutative. Furthermore, labels are public and their computation no longer depends on the secret key.

Definition 40 (TACO Syntax)**.** *Let $\mathcal{M}$ be the message space, $\mathcal{C}$ the space of ciphertexts, $\mathcal{L}$ the space of labels, $\mathcal{K}$ the space of keys and $\mathcal{I}$ the space of identifiers. Furthermore, let Φ be a set of plaintext operations. Each $\varphi \in \Phi$ has a fixed number of parameters p_φ such that φ maps a p_φ-dimensional tuple $(m_1, \ldots, m_{p_\varphi})$ with $m_j \in \mathcal{M}$ to one message $\hat{m} \in \mathcal{M}$. A trusted authenticated ciphertext operations (TACO) scheme is a tuple of PPT algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Der}, \mathsf{Dec})$ *such that:*

- *The* key-generation algorithm Gen *takes the security parameter 1^λ as input and outputs a secret key $sk \in \mathcal{K}$.*
- *The* encryption algorithm Enc *takes a secret key sk, a plaintext message $m \in \mathcal{M}$ and an identifier $i \in \mathcal{I}$ as input and outputs a ciphertext $c \in \mathcal{C}$.*
- *The* evaluation algorithm Eval *takes a function $\varphi \in \Phi$ and a p_φ-dimensional tuple $(c_1, \ldots, c_{p_\varphi})$ with $c_j \in \mathcal{C}$ as input and outputs a ciphertext $\hat{c} \in \mathcal{C}$ or $\perp$ on authentication error.*
- *The deterministic* label derivation algorithm Der *takes either*
  - *an identifier $i \in \mathcal{I}$ or*
  - *a function $\varphi \in \Phi$ and a p_φ-dimensional tuple $(l_1, \ldots, l_{p_\varphi})$ with $l_j \in \mathcal{L}$*

  as input and outputs a label $\hat{l} \in \mathcal{L}$.
- *The deterministic* decryption algorithm Dec *takes a secret key sk, a ciphertext $c \in \mathcal{C}$ and a label $l \in \mathcal{L}$ as input and outputs a plaintext message $m \in \mathcal{M}$ or $\perp$ on decryption error.*

In order for a TACO scheme to be correct, the decryption must be successful and yield the expected value also after multiple evaluations. For this purpose we introduce the following definitions:

Definition 41 (TACO Partial Correctness)**.** *Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Der}, \mathsf{Dec})$ be a TACO scheme. We say that Π is* partially correct *if for all $m \in \mathcal{M}$ and all $i \in \mathcal{I}$ it holds that*

$$\mathsf{Dec}(sk, \mathsf{Enc}(sk, m, i), \mathsf{Der}(i)) = m.$$

Definition 42 (TACO Ciphertext Validity)**.** *Let $c \in \mathcal{C}$ be a ciphertext. We say c is* valid *if and only if*

$$\exists l \in \mathcal{L} : \mathsf{Dec}(sk, c, l) \neq \perp.$$

The corresponding $l \in \mathcal{L}$, for which $\mathsf{Dec}(sk, c, l) \neq \perp$ is denoted as a valid *label for c.*

Definition 43 (TACO Correctness)**.** *Let* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Der}, \mathsf{Dec})$ *be a partially correct TACO scheme. We say that* Π *is* correct *if for any secret key* $sk \leftarrow \mathsf{Gen}(1^\lambda)$*, any function* $\varphi \in \Phi$*, any* p_φ*-dimensional tuple of ciphertexts* $C := (c_1, \ldots, c_{p_\varphi})$ *with* $c_j \in \mathcal{C}$ *and* c_j valid *and any* p_φ*-dimensional tuple of labels* $L := (l_1, \ldots, l_{p_\varphi})$ *with* $l_j \in \mathcal{L}$ *and* l_j valid for c_j*, for the* p_φ*-dimensional tuple of plaintexts* $M := (m_1, \ldots, m_{p_\varphi})$ *with* $m_j = \Pi.\mathsf{Dec}(sk, c_j, l_j)$ *it holds that*

$$\mathsf{Dec}(sk, \mathsf{Eval}(\varphi, C), \mathsf{Der}(\varphi, L)) = \varphi(M).$$

5.5.2 Security Definitions

Indistinguishability of a TACO scheme is defined like HASE-IND-CPA (Definition 38) except that the adversary is provided with access to an evaluation oracle rather than the evaluation key. Unforgeability of a TACO scheme is based on the unforgeable encryption definition (Definition 15). Essentially, the unforgeability adversary wins by producing two ciphertexts which decrypt to different messages under the same label.

Definition 44 (TACO-IND-CPA)**.** *A TACO scheme* Π *has* indistinguishable encryptions under a chosen-plaintext attack*, or is* CPA-secure*, if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{A},\Pi}(\lambda) := \left|\Pr\left[\text{ExpTACO}^{\text{IND-CPA}}_{\mathcal{A},\Pi}(\lambda) = 1\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda)$$

The experiment is defined as follows:

```
$\text{ExpTACO}^{\text{IND-CPA}}_{\mathcal{A},\Pi}(\lambda)$
$sk \leftarrow \Pi.\mathsf{Gen}(1^\lambda)$
$(m_1, m_2, i_1, i_2, \mathsf{st}) \leftarrow \mathcal{A}^{\Pi.\mathsf{Enc}(sk,\cdot,\cdot),\Pi.\mathsf{Eval}(sk,\cdot,\cdot)}(1^\lambda)$
$b \leftarrow_\$ \{0,1\}$
$c \leftarrow \Pi.\mathsf{Enc}(sk, m_b, i_b)$
$b' \leftarrow \mathcal{A}^{\Pi.\mathsf{Enc}(sk,\cdot,\cdot),\Pi.\mathsf{Eval}(sk,\cdot,\cdot)}(1^\lambda, c, \mathsf{st})$
return $b = b'$
```

Definition 45 (TACO-UF-CPA)**.** *A TACO scheme* Π *is* unforgeable under a chosen-plaintext attack*, or just* unforgeable*, if for all PPT adversaries* $\mathcal{A}$ *and all functions* $\varphi \in \Phi_\Pi$ *there is a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\mathsf{Adv}^{\text{UF-CPA}}_{\mathcal{A},\Pi}(\lambda) := \Pr\left[\text{ExpTACO}^{\text{UF-CPA}}_{\mathcal{A},\Pi,\varphi}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

with the experiment defined as follows:

```
$\text{ExpTACO}^{\text{UF-CPA}}_{\mathcal{A},\Pi,\varphi}(\lambda)$
$I := \{\}$
$sk \leftarrow \Pi.\mathsf{Gen}(1^\lambda)$
$(c_1, c_2, l) \leftarrow \mathcal{A}^{E(sk,\cdot,\cdot),\Pi.\mathsf{Eval}(sk,\cdot,\cdot)}(1^\lambda)$
$m_1 := \Pi.\mathsf{Dec}(sk, c_1, l)$
$m_2 := \Pi.\mathsf{Dec}(sk, c_2, l)$
return $m_1 \neq \perp \wedge m_2 \neq \perp \wedge m_1 \neq m_2$
```

```
$E(sk, m, i)$
if $i \in I$ then
    return $\perp$
else
    $I := I \cup \{i\}$
    $c \leftarrow \Pi.\mathsf{Enc}(sk, m, i)$
    return $c$
```

5.5.3 Construction

Construction 4 (TACO)**.** *Define the message space $\mathcal{M} := \{0,1\}^*$, the ciphertext space $\mathcal{C} := \{0,1\}^*$, the label space $\mathcal{L} := \{0,1\}^\lambda$ and the space of identifiers $\mathcal{I} := \{0,1\}^*$. Let $SE = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric encryption scheme with message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. Define the key space $\mathcal{K}$ as the space of keys output by $SE.\mathsf{Gen}$. For a hash function $\Omega = (\mathsf{Gen}, H)$ obtain $k \leftarrow \Omega.\mathsf{Gen}(1^\lambda)$. Furthermore, let Φ be a set of plaintext operations and let $\mathtt{id} : \Phi \mapsto \{0,1\}^*$ be an injective function. Construct a TACO scheme using the following PPT algorithms:*

- Gen*: on input 1^λ obtain and output a symmetric encryption secret key*

$$sk \leftarrow SE.\mathsf{Gen}(1^\lambda).$$

- Enc*: on input a secret key sk and a message $m \in \mathcal{M}$, output the ciphertext*

$$c \leftarrow SE.\mathsf{Enc}(sk, m \| \mathsf{Der}(i)).$$

- Eval*: on input a secret key sk, an operation $\varphi \in \Phi$ and a p_φ-dimensional tuple of ciphertexts $(c_1, \ldots, c_{p_\varphi})$.*
  For $j \in [1, p_\varphi]$:
  - *Compute $d_j := SE.\mathsf{Dec}(sk, c_j)$.*
  - *Parse $d_j = m_j \| l_j$.*

  If any $d_j = \perp$, then return $\perp$. Otherwise compute
  - $m' := \varphi\left((m_1, \ldots, m_{p_\varphi})\right)$
  - $l' := \mathsf{Der}\left(\varphi, (l_1, \ldots, l_{p_\varphi})\right)$
  - $\hat{c} \leftarrow SE.\mathsf{Enc}\left(sk, m' \| l'\right)$

  and output the ciphertext $\hat{c}$.

- Der*:*
  - *on input a secret key sk and an identifier $i \in \mathcal{I}$, output the label*

$$\hat{l} := H(k, i \| 0).$$

  - *on input a secret key sk, a function $\varphi \in \Phi$ and a p_φ-dimensional tuple of labels $(l_1, \ldots, l_{p_\varphi})$, output the label*

$$\tilde{l} := H\left(k, \mathtt{id}(\varphi) \| l_1 \| \ldots \| l_{p_\varphi} \| 1\right).$$

- Dec*: on input a secret key sk, a ciphertext $c \in \mathcal{C}$ and label $l \in \mathcal{L}$, compute $d := SE.Dec(sk, c)$ and parse $d = m' \| l'$. If $d = \perp$ or $l \neq l'$, then output $\perp$. Otherwise output m'.*

5.5.4 Security Reductions

Theorem 6 (TACO-IND-CPA). *Let* Π *be Construction 4 and* SE *its symmetric encryption scheme. If* SE *is CCA-secure (Definition 6), then* Π *is* TACO-IND-CPA *secure.*

Proof. Let Π and SE as before. Note that if SE is CCA-secure, it is also CPA-secure (Definition 5).

Assume there exists an adversary winning the TACO-IND-CPA game with non-negligible probability. By definition, the adversary can then distinguish for chosen messages $m_1, m_2 \in \mathcal{M}$ and identifiers $i_1, i_2 \in \mathcal{I}$ the ciphertext

$$c_1 \leftarrow \Pi.\mathsf{Enc}(sk, m_1, i_1) \overset{\text{Def}}{=} SE.\mathsf{Enc}(sk, m_1 \| \mathsf{Der}(i_1))$$

from

$$c_2 \leftarrow \Pi.\mathsf{Enc}(sk, m_2, i_2) \overset{\text{Def}}{=} SE.\mathsf{Enc}(sk, m_2 \| \mathsf{Der}(i_2))$$

with non-negligible probability. In case the adversary performed no evaluation oracle queries, this is a contradiction to IND-CPA security of SE.

To show that the evaluation oracle provides only negligible advantage to the adversary, we simulate it as an attacker on the IND-CCA property of the SE scheme. The IND-CCA property of the SE scheme provides indistinguishability with additional access to a decryption oracle: Assume the adversary on TACO wants to perform an evaluation φ on ciphertexts $C = (c_1, \ldots, c_{p_\varphi})$. In case C does not include the challenge ciphertext c, the evaluation can be simulated by decrypting each ciphertext using the $SE.\mathsf{Dec}$ oracle, calculating φ on the results, and encrypting with the encryption oracle again. In case the adversary wants to perform an evaluation φ on a tuple including the challenge ciphertext c, instead of computing $SE.\mathsf{Dec}(sk, c)$, we use m_b, i with $b \leftarrow\!\!\$\, \{0,1\}$ for the calculation of φ. If the attacker can distinguish the simulation from a honest oracle with non-negligible probability, he can again distinguish $SE.\mathsf{Enc}(m_1')$ from $SE.\mathsf{Enc}(m_2')$ for at least two $m_1', m_2' \in \mathcal{M}$ with non-negligible probability. Since the attacker can only perform a number of evaluations polynomial in λ, we get the security as induction over the number of evaluations.

□

Theorem 7 (TACO-UF-CPA). *Let* Π *be Construction 4,* SE *its symmetric encryption scheme and* $\Omega = (\mathsf{Gen}, H)$ *its hash function. If* SE *is unforgeable (Definition 15) and* Ω *is collision-resistant (Definition 13), then* Π *is* TACO-UF-CPA *secure.*

Proof. Let Π, SE and Ω as before. The instructions defined in the TACO-UF-CPA game lead directly to the following equalities:

$$\begin{aligned}
&\mathsf{Adv}^{\text{UF-CPA}}_{\mathcal{A},\Pi}(\lambda) \\
&= \Pr[m_1 \neq \perp \wedge m_2 \neq \perp \wedge m_1 \neq m_2] \\
&= \Pr[\Pi.\mathsf{Dec}(sk, c_1, l) \neq \perp \wedge \Pi.\mathsf{Dec}(sk, c_2, l) \neq \perp \wedge m_1 \neq m_2] \\
&= \Pr[SE.\mathsf{Dec}(sk, c_1) \neq \perp \wedge l_1 = l \wedge SE.\mathsf{Dec}(sk, c_2) \neq \perp \wedge l_2 = l \wedge m_1 \neq m_2] \\
&\leq \Pr[l_1 = l_2 \wedge SE.\mathsf{Dec}(sk, c_1) \neq \perp \wedge SE.\mathsf{Dec}(sk, c_2) \neq \perp \wedge m_1 \neq m_2] \qquad (5.6)
\end{aligned}$$

Here, l_1 and l_2 are the parsed labels of the back part of the $SE.\mathsf{Dec}$ decrypted ciphertexts. Applying the definition of conditional probability results in:

$$\begin{aligned}&\mathsf{Adv}^{\text{UF-CPA}}_{\mathcal{A},\Pi}(\lambda)\\ \leq &\Pr[l_1 = l_2 \mid m_1 \neq m_2 \wedge SE.\mathsf{Dec}(sk, c_1) \neq \perp \wedge SE.\mathsf{Dec}(sk, c_2) \neq \perp]\end{aligned} \tag{5.7}$$

Since the underlying encryption scheme is assumed to be unforgeable, in order for $SE.\mathsf{Dec}(sk, c_1) \neq \perp \wedge SE.\mathsf{Dec}(sk, c_2) \neq \perp$ to hold with non-negligible probability, the ciphertexts must have been created using the secret key. This implies that the ciphertexts were created by either the evaluation oracle or the encryption oracle. For the evaluation not to result in $\perp$, it must be performed on valid ciphertexts. Let $\Pr[S_n]$ be the probability in inequality 5.7, where n is the number of evaluations an adversary performs on valid ciphertexts to create the ciphertexts c_1 and c_2 before passing them to the challenger. We show the security by an induction proof over n.

- Claim: $\Pr[S_n]$ is negligible $\forall n \in \mathbb{N}$ with n polynomial in λ.
- First base case for $n = 0$: No evaluations have been performed such that:

  $$\begin{aligned}\Pr[S_0] = \Pr\ &[l_1 = l_2 \mid m_1 \neq m_2\\ &\quad \wedge c_1 = E(sk, m_1, i_1) \wedge c_2 = E(sk, m_2, i_2)]\\ = \Pr\ &[H(k, i_1\|0) = H(k, i_2\|0) \mid m_1 \neq m_2\\ &\quad \wedge c_1 = E(sk, m_1, i_1) \wedge c_2 = E(sk, m_2, i_2)]\end{aligned}$$

  Since the encryption oracle does not allow to encrypt different messages with the same identifier, it must hold that $i_1 \neq i_2$ which leads to

  $$\Pr[S_0] = \Pr[H(k, i_1\|0) = H(k, i_2\|0) \mid i_1 \neq i_2]$$

  This probability is negligible due to the assumption that Ω is collision-resistant.
- Second base case for $n = 1$: Without loss of generality let c_1 be the ciphertext created by an evaluation of an operation φ and valid ciphertexts $(c_{1,1}, \ldots, c_{1,p_\varphi})$ with corresponding labels $(l_{1,1}, \ldots, l_{1,p_\varphi})$. Then:

  $$\begin{aligned}\Pr[S_1] = \Pr\ &[l_1 = l_2 \mid m_1 \neq m_2\\ &\quad \wedge c_1 = \mathsf{Eval}(sk, \varphi, (c_{1,1}, \ldots, c_{1,p_\varphi})) \wedge c_2 = E(sk, m_2, i_2)]\\ = \Pr\ &\left[H(k, \mathtt{id}(\varphi)\|l_{1,1}\| \ldots \|l_{1,p_\varphi}\|1) = H(k, i_2\|0)\right]\end{aligned}$$

  This probability is negligible due to the assumption that Ω is collision-resistant.
- Step case for $n \rightsquigarrow n + 1$: Assume the claim is true for $0 \leq n^* \leq n$. If only one challenger ciphertext is created by evaluations, then the proof in the second base case applies. Otherwise for $j \in \{0, 1\}$ let c_j be the ciphertext created by an evaluation of operation φ_j and valid ciphertexts

$(c_{j,1},\ldots,c_{j,p_{\varphi_j}})$ with corresponding labels $(l_{j,1},\ldots,l_{j,p_{\varphi_j}})$ and corresponding messages $(m_{j,1},\ldots,m_{j,p_{\varphi_j}})$. Then:

$$\begin{aligned}
\Pr[S_{n+1}] &= \Pr\,[l_1 = l_2 \mid m_1 \neq m_2 \\
&\qquad \wedge \forall j \in \{0,1\} : c_j = \mathsf{Eval}(sk, \varphi_j, (c_{j,1},\ldots,c_{j,p_{\varphi_j}})) \\
&\qquad \wedge SE.\mathsf{Dec}(sk,c_1) \neq \bot \wedge SE.\mathsf{Dec}(sk,c_2) \neq \bot] \\
&= \Pr\,[H(k, \texttt{id}(\varphi_1) \| l_{1,1} \| \ldots \| l_{1,p_{\varphi_1}} \| 1) = \\
&\qquad H(k, \texttt{id}(\varphi_2) \| l_{2,1} \| \ldots \| l_{2,p_{\varphi_2}} \| 1) \mid m_1 \neq m_2 \\
&\qquad\quad \wedge SE.\mathsf{Dec}(sk,c_1) \neq \bot \wedge SE.\mathsf{Dec}(sk,c_2) \neq \bot] \\
&= \Pr\,[H(\ldots) = H(\ldots) \wedge (\varphi_1 \neq \varphi_2 \vee \exists j : l_{1,j} \neq l_{2,j}) \mid m_1 \neq m_2 \\
&\qquad \wedge SE.\mathsf{Dec}(sk,c_1) \neq \bot \wedge SE.\mathsf{Dec}(sk,c_2) \neq \bot] \qquad (5.8) \\
&+ \Pr\,[H(\ldots) = H(\ldots) \wedge (\varphi_1 = \varphi_2 \wedge \forall j : l_{1,j} = l_{2,j}) \mid m_1 \neq m_2 \\
&\qquad \wedge SE.\mathsf{Dec}(sk,c_1) \neq \bot \wedge SE.\mathsf{Dec}(sk,c_2) \neq \bot] \qquad (5.9)
\end{aligned}$$

Probability (5.8) is negligible since the input to H differs and Ω is collision-resistant. We explore the Probability (5.9) in more detail by considering its conditions.

$$\begin{aligned}
& m_1 \neq m_2 \\
&\Rightarrow SE.\mathsf{Dec}(sk, \mathsf{Eval}(sk, \varphi_1, (c_{1,1},\ldots,c_{1,p_{\varphi_1}}))) \\
&\qquad \neq SE.\mathsf{Dec}(sk, \mathsf{Eval}(sk, \varphi_2, (c_{2,1},\ldots,c_{2,p_{\varphi_2}}))) \\
&\Rightarrow \varphi_1(SE.\mathsf{Dec}(sk,c_{1,1}),\ldots,SE.\mathsf{Dec}(sk,c_{1,p_{\varphi_1}})) \\
&\qquad \neq \varphi_2(SE.\mathsf{Dec}(sk,c_{2,1}),\ldots,SE.\mathsf{Dec}(sk,c_{2,p_{\varphi_2}})) \\
&\Rightarrow \varphi_1(SE.\mathsf{Dec}(sk,c_{1,1}),\ldots,SE.\mathsf{Dec}(sk,c_{1,p_{\varphi_1}})) \\
&\qquad \neq \varphi_1(SE.\mathsf{Dec}(sk,c_{2,1}),\ldots,SE.\mathsf{Dec}(sk,c_{2,p_{\varphi_1}})) \\
&\Rightarrow \exists j^* : SE.\mathsf{Dec}(sk,c_{1,j^*}) \neq SE.\mathsf{Dec}(sk,c_{2,j^*}) \qquad (5.10)
\end{aligned}$$

We choose such an j^* fulfilling inequality (5.10) and obtain:

$$\begin{aligned}
\Pr[S_{n+1}] &= \Pr\,[H(\ldots) = H(\ldots) \wedge \varphi_1 = \varphi_2 \wedge \forall j : l_{1,j} = l_{2,j} \mid m_1 \neq m_2 \\
&\qquad \wedge SE.\mathsf{Dec}(sk,c_1) \neq \bot \wedge SE.\mathsf{Dec}(sk,c_2) \neq \bot] + negl(\lambda) \\
&\leq \Pr\,[H(\ldots) = H(\ldots) \wedge \varphi_1 = \varphi_2 \wedge l_{1,j^*} = l_{2,j^*} \mid m_1 \neq m_2 \\
&\qquad \wedge SE.\mathsf{Dec}(sk,c_1) \neq \bot \wedge SE.\mathsf{Dec}(sk,c_2) \neq \bot] + negl(\lambda) \\
&\leq \Pr\,[l_{1,j^*} = l_{2,j^*} \mid m_1 \neq m_2 \\
&\qquad \wedge SE.\mathsf{Dec}(sk,c_1) \neq \bot \wedge SE.\mathsf{Dec}(sk,c_2) \neq \bot] + negl(\lambda) \\
&= \Pr\,[l_{1,j^*} = l_{2,j^*} \mid m_1 \neq m_2 \\
&\qquad \wedge SE.\mathsf{Dec}(sk,c_1) \neq \bot \wedge SE.\mathsf{Dec}(sk,c_2) \neq \bot \\
&\qquad \wedge SE.\mathsf{Dec}(sk,c_{1,j^*}) \neq SE.\mathsf{Dec}(sk,c_{2,j^*})] + negl(\lambda) \\
&\leq \Pr\,[l_{1,j^*} = l_{2,j^*} \mid SE.\mathsf{Dec}(sk,c_1) \neq \bot \\
&\qquad \wedge SE.\mathsf{Dec}(sk,c_2) \neq \bot \wedge m_{1,j^*} \neq m_{2,j^*}] + negl(\lambda)
\end{aligned}$$

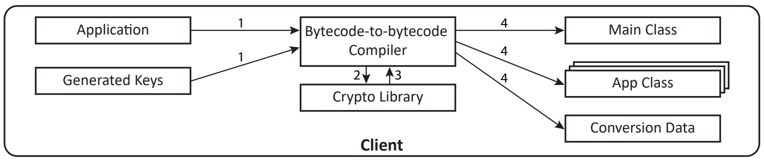


Figure 5.5: Application transformation during DFAuth setup phase.

Moreover it holds:

$$\begin{aligned}
&SE.\mathsf{Dec}(sk, c_1) \neq \perp \wedge SE.\mathsf{Dec}(sk, c_2) \neq \perp \\
&\Rightarrow SE.\mathsf{Dec}(\mathsf{Eval}(sk, \varphi_1, (c_{1,1}, \ldots, c_{1,p_{\varphi_1}}))) \neq \perp \\
&\qquad \wedge SE.\mathsf{Dec}(\mathsf{Eval}(sk, \varphi_2, (c_{2,1}, \ldots, c_{2,p_{\varphi_2}}))) \neq \perp \\
&\Rightarrow SE.\mathsf{Dec}(c_{1,j^*}) \neq \perp \wedge SE.\mathsf{Dec}(c_{2,j^*}) \neq \perp
\end{aligned}$$

$$\begin{aligned}
\Pr[S_{n+1}] \leq \Pr\ &[l_{1,j^*} = l_{2,j^*} \mid SE.\mathsf{Dec}(c_{1,j^*}) \neq \perp \wedge SE.\mathsf{Dec}(c_{2,j^*}) \neq \perp \\
&\wedge m_{1,j^*} \neq m_{2,j^*}] + negl(\lambda) \\
= \Pr&[S_{n^*}] + negl(\lambda) \qquad , n^* \leq n
\end{aligned}$$

This probability is negligible according to the induction hypothesis which proofs that $\forall n \in \mathbb{N}\ \Pr[S_n]$ is negligible. Because the number of evaluations an adversary can perform n is polynomial in λ, $\mathsf{Adv}_{\mathcal{A},\Pi}^{\text{UF-CPA}}(\lambda)$ is negligible.

$\square$

5.6 Implementation

In this section, we present details of an implementation in Java used in our experiments. Recall from Section 5.2.1 that our scenario considers a trusted *client* and an untrusted cloud *server* which has a *trusted module*. Also recall that we distinguish two phases of the outsourced computation: *setup* and *runtime*.

5.6.1 Setup Phase

The setup phase is divided into two parts, *compilation* and *deployment*, as described below. An overview is provided in Figure 5.5.

Compilation. First, the client translates any Java bytecode program to a bytecode program running on encrypted data. To start, the client generates a set of cryptographic keys. It then uses our bytecode-to-bytecode compiler to transform an application (in the form of Java bytecode) using the generated keys (1). Our compiler is based on Soot, a framework for analyzing and transforming Java applications [Lam+11]. It uses our DFAuth crypto library to encrypt program constants and choose variable labels (2–3).

The DFAuth crypto library contains implementations of all required cryptographic algorithms, including our own from Sections 5.4 and 5.5. It implements the PRF used for authentication labels as HMAC-SHA256 [EH06]. For the group operations in Multiplicative HASE we use MPIR [Mpi] for large integer arithmetic. Additive HASE operates on the elliptic curve group provided by `libsodium` [Lib]. The Gen method of Additive HASE has as parameters the number of ciphertext components and the number of bits per component. From these, it deterministically derives a set of t primes. The Additive HASE Dec method computes the discrete logarithms using exhaustive search with a fixed set of precomputed values. To ensure the efficiency of Additive HASE decryption, the compiler inserts trusted-module invocations into the program that decrypt and re-encrypt ciphertexts. These re-encryptions result in modulo reductions in the exponents (cf. Construction 3), thus preventing excessive exponent growth and ensuring an efficient decryption. The frequency of these invocations can be defined by the application. We demonstrate the efficacy of these re-encryptions in Section 5.7.2. For HASE, our compiler converts floating-point plaintexts to a fixed-point representation using an application-defined scaling factor. It also transforms the calculations to integer values, whereby the scaling factors are considered when appropriate. The resulting value is transformed back to floating-point after decryption. For the symmetric encryption scheme in TACO we use the Advanced Encryption Standard [Aes] in Galois/counter mode (AES-GCM). AES-GCM is an authenticated encryption scheme and as such is both CCA-secure and unforgeable (cf. Definition 16).

Finally, the compiler performs the transformation described in Section 5.3 and outputs a *main class* containing the program start code, multiple *app classes* containing the remaining code and *conversion data* (e.g., labels and comparison data) (4).

Deployment. Second, the client deploys the app classes at the cloud server and securely loads the generated cryptographic keys and conversion data into the trusted module. We implemented the trusted module using an Intel SGX enclave (cf. Section 3.4.1). SGX is well suited for our implementation because it provides isolated program execution (including strong memory encryption) and remote attestation (including a secure communication channel). The client uses remote attestation to prepare the enclave. It verifies the correct creation of the DFAuth trusted module enclave in the remote system and the correct setup of the crypto library. At the same time, the client establishes a secure communication channel with the remote enclave, over which the sensitive conversion data is loaded. The secure channel provides confidentiality and authenticity. It protects the communication between the trusted client and the trusted enclave against the untrusted part of the server as well as any other attackers on the network. We also emphasize that SGX's hardware protections protect cryptographic keys and conversion data on the server from access by any software except our DFAuth enclave.

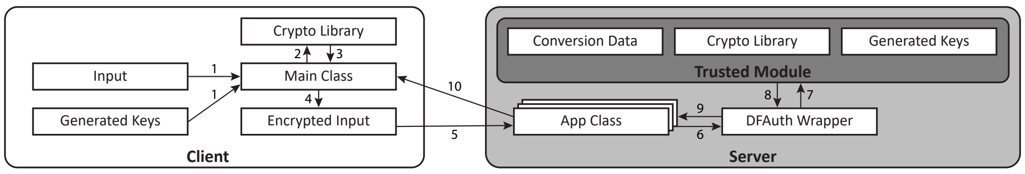


Figure 5.6: Application execution during DFAuth runtime phase.

5.6.2 Runtime Phase

To run the program, the client executes the main class which triggers the remote program execution at the cloud server (see Figure 5.6). The main class encrypts the *program input* (for this run of the program) with the generated keys (for the entire setup of the program) using the crypto library (1–4). The main class passes the *encrypted input* to the app classes on the cloud server (5). The app classes operate on encrypted data and do not have any additional protection. They invoke the *DFAuth wrapper* for operations on homomorphic ciphertexts and re-encryption or comparison requests (6). The wrapper hides the specific homomorphic encryption schemes and trusted module implementation details from the app classes, such that it is feasible to run the same program using different encryption schemes or trusted modules. It forwards re-encryption and comparison requests to the trusted module and passes the answers back to the application (7–9). Once the app classes finish their computation, they send an encrypted result (including an authentication label) back to the client (10). The client verifies the authentication label to the one computed by our compiler.

The task of the trusted module during runtime is to receive re-encryption and comparison requests, determine whether they are legitimate and answer them if they are. It bundles cryptographic keys, authentication labels and required parts of the crypto library inside a trusted area, shielding it from unauthorized access. The DFAuth wrapper enables to potentially select different trusted modules based on the client's requirements and their availability at the cloud server. Besides Intel SGX enclaves, one can implement a trusted module using a hypervisor or calling back to the client for sensitive operations. However, alternative implementations would involve making slightly different trust assumptions.

SGX's secure random number generator provides the randomness required during encryption. A restriction of the current generation of Intel SGX is the limited size of its isolated memory (cf. Section 3.4.1). It only provides about 96 MB for code and data and thus enforces an upper bound on the number of precomputed discrete logarithm values used to speedup Additive HASE. The available memory can be used optimally with a careful selection of CRT parameters.

TACO evaluation, HASE re-encryption and comparison requests have to be implemented inside the trusted module. We display the conversion routines (implemented in an SGX enclave in our case) for conversion to Multiplicative HASE and comparison in Listings 5.3 and 5.4. The conversion routine to Additive HASE is similar to the one for Multiplicative HASE in Listing 5.3 with the roles of the encryption schemes switched. The comparison of two encrypted values

Listing 5.3: Conversion to Multiplicative HASE

```
convertToMul(x, "x") {
    label = labelLookup("x");
    y = Dec(K, x, label);
    if (y == fail)
        stop;
    id = idLookup("x");
    return Enc(K, y, id);
}
```

Listing 5.4: Conversion to comparison

```
convertToCmp(x, y, "x") {
    label = labelLookup("x");
    x1 = Dec(K, x, label);
    if (x1 == fail)
        stop;
    if (y == null) {
        param = paramLookup("x");
        switch (param.type) {
            case EQ:
                return (x1 == param.const);
            case GT:
                return (x1 > param.const);
            case GTE:
                ...
        }
    } else {
        label = labelLookup("y");
        y1 = Dec(K, y, label);
        if (y1 == fail)
            stop;
        ...
    }
}
```

is similar to the comparison of one to a constant in Listing 5.4. Similar to the call `labelLookup`, which retrieves labels from conversion data stored inside the trusted module, `idLookup` and `paramLookup` retrieve identifiers for encryption and parameters for comparison from the conversion data.

5.7 Evaluation

In this section, we present the evaluation results collected in various experiments. Our experiments are grouped into two series consisting of two experiments each. The first series focusses on DFAuth with HASE. The second series focusses on more complex applications and a comparison of HASE and TACO.

Experiments of the first series are performed on an SGX-capable Intel Core i7-6700 CPU with 64 GB RAM running Windows 10. In our first experiment (presented in Section 5.7.1), we apply DFAuth to the checkout (shopping cart) component of a secure sales application, which we developed ourselves. In the second experiment (presented in Section 5.7.2), we benchmark the Additive HASE construction for large numbers of additions.

Experiments of the second series are performed in the Microsoft Azure Cloud using Azure Confidential Computing. We use `Standard_DC4s` VM instances which run Ubuntu Linux 18.04 and have access to 4 cores of an SGX-capable Intel Xeon E-2176G CPU and 16 GB RAM. In the third experiment (presented in Section 5.7.3), we apply DFAuth to an existing neural network program enabling secure neural network evaluation in the cloud. In the fourth experiment (presented in Section 5.7.4), we use DFAuth to protect sensitive data processed by a smart charging scheduler for electric vehicles. In both experiments, we separately evaluate DFAuth with HASE and DFAuth with TACO. We collect the running time inside and outside the trusted module as well as the number of operations performed inside and outside the trusted module.

In all experiments, we aim for a security level equivalent to 80 bits of symmetric encryption security. We use the 1536-bit MODP Group from RFC3526 [KK03] as the underlying group in Multiplicative HASE. The `libsodium` [Lib] elliptic curve group used by Additive HASE even provides a security level of 128 bits [Ber06]. A key length of 128 bits is used for the AES-GCM symmetric encryption scheme in TACO.

5.7.1 Secure Sales Application

In this experiment, we consider the checkout component of a secure sales application running on an untrusted server. When a client checks out a shopping cart, the server is tasked with summing up the encrypted prices of all items in the cart. Additionally, discounts need to be applied when the sum exceeds certain thresholds. We aim to protect the rebate structure of the client, i.e. the thresholds when discounts are applied and the percentage of the discount. This is important in outsourced sales applications, because an attacker, e.g. a co-residing competitor, could try to infer the rebate structure in order to gain an unfair advantage in the market.

The purpose of this experiment is twofold. First, this experiment serves as a performance test for the implementation of our HASE constructions. Second, this experiment serves as an example of the larger class of non-linear functions. While the secrets in this illustrative example can be inferred fairly easily, with growing number of parameters inference gets harder. Note that the constants in almost any computation can be inferred by repetitive queries. It is a question of costs whether it is economically worthwhile to do it. Also note that in a business to business sales application, almost each customer has its own confidential rebate structure.

The transformed program performs control-flow decisions comparing the sum of encrypted inputs to the encrypted thresholds of the rebate structure. To a control-flow observer each such decision reveals a small amount of information about the relationship between inputs and threshold constants. After observing the result of sufficiently many such decisions, an attacker may be capable of inferring the confidential threshold values. Note that the attacker does not learn additional information about the output and most importantly the discounts granted.

Experimental Setup. We implemented the described checkout component and applied DFAuth with HASE to it. If the sum exceeds the value of \$250, our implementation grants a discount of 5%. If the sum exceeds the value of \$500, we grant a total discount of 10%.

In order to evaluate the performance of the original (*plaintext*) and the DFAuth variant of the program, we built shopping carts of sizes $\{1, 10, \ldots, 100\}$. Prices were taken uniformly at random from the interval $[0.01, 1000.00]$. For each cart a plaintext and an encrypted variant is stored at the untrusted server.

For each of the two program variants, the total running time of code executing at the untrusted server was measured. This time includes reading the corresponding cart data from disk, summing up the prices of all cart items and granting discounts where applicable. For the DFAuth variant, we also collected the number of operations performed on encrypted data inside and outside of the trusted module, as well as the time spent invoking and inside the trusted module. Our measurements do not include the setup phase, because it is only a one-time overhead that amortizes over multiple runs. We also do not include network latency in our measurements, since the difference in communication between a program running on plaintext and a program running on encrypted data is very small.

Evaluation Results. There are three cases for the control flow in the secure sales application:

1. The sum of all item prices neither reaches the first threshold nor the second threshold. In this case, the sum of the prices is compared to two different threshold constants.

2. The sum of all prices reaches the larger threshold. In this case, the sum is compared to one threshold constant and needs to be converted to Multiplicative HASE before being multiplied with the respective discount constant.

3. The sum of all prices reaches the lower threshold, but not the larger threshold. In this case, the sum is compared to two threshold constants and needs to be converted to Multiplicative HASE before being multiplied with the respective discount constant.

Figure 5.7 presents the mean running time of 100 runs of the experiment described above for Case 3. We can see that, even for large-sized carts containing 100 items, DFAuth only increases the program running time by a factor of 11.5

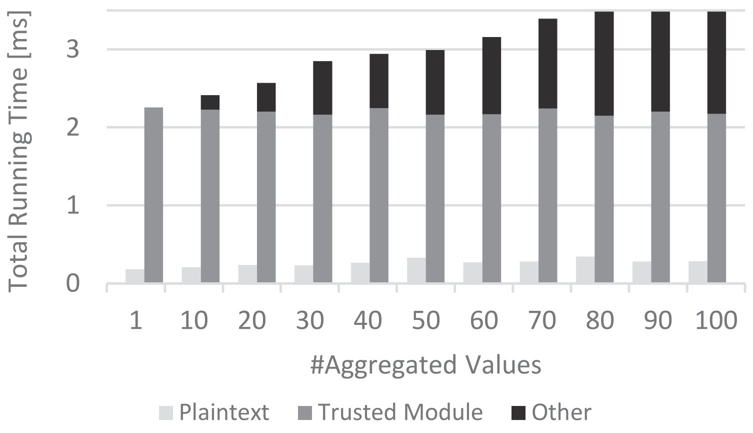


Figure 5.7: Mean running time [ms] of the original (left) and DFAuth (right) variants of the shopping cart program as a function of the cart size for Case 3.

Operation	**Type**	**# Ops in Case 1/2/3**
Homomorphic Addition	HASE	9/9/9
Homomorphic Multiplication	HASE	0/1/1
Total	HASE	9/10/10
Additive to Multiplicative Conversion	SGX	0/1/1
Comparison to Constant	SGX	2/1/2
Total	SGX	2/2/3

Table 5.2: Number of untrusted (HASE) and trusted (SGX) operations in the Secure Sales Application experiment for shopping cart size 10.

on average. Assuming a round-trip latency of 33 ms, which can be considered realistic according to [Net], the total round-trip time (consisting of round-trip latency and program running time) increases only by a factor of 1.08 on average. Most importantly, the absolute running time values are sufficiently low for practical deployment of online computation.

From Figure 5.7 we can also see that a significant portion of the total running time is spent inside (or invoking) the trusted module. On the one hand, this shows that a more efficient trusted module implementation would significantly decrease the total running time of the application. On the other hand, it suggests that we execute more instructions inside the trusted module than outside, contradicting our basic idea of a reduced execution inside the trusted module. However, Table 5.2, which reports the number of operations performed on encrypted data inside and outside of the trusted module, shows that this is not the case. Even for a shopping cart containing only 10 items, there are 9 to 10 untrusted HASE operations, but only 1 to 3 trusted operations. While the number of trusted operations is independent of the cart size, the number of untrusted operations is approximately linear in the cart size, i.e., an even larger cart would further increase the fraction of untrusted operations. We refer to the next section for a demonstration of Additive HASE's scalability.

5.7.2 Additive HASE Benchmark

The decryption algorithm of Additive HASE (Construction 3) has to compute a discrete logarithm for each ciphertext component (after Elgamal decryption). Since each homomorphic addition can increase the bit length of exponents by 1, a large amount of homomorphic additions can make decryption exponentially costlier (or impossible, assuming only a lookup table with precomputed logarithms), despite the use of the CRT approach provided by Hu et al. [HMS12]. In this experiment, we demonstrate that re-encryptions inserted by the DFAuth compiler are an effective measure for preventing excessive exponent growth and ensuring an efficient decryption.

Experimental Setup. Throughout this experiment, we use a CRT decomposition involving 13 different 5-bit primes. These parameters were chosen such that we can represent 64-bit integer values. In the trusted module a lookup table containing 2^{18} precomputed discrete logarithms was generated. We measure the running time of the decryption algorithm when applied to a ciphertext resulting from the homomorphic evaluation of n ciphertexts. For each n we consider two variants of the experiment: one without re-encryptions, the other with re-encryptions performed after every 4000 homomorphic evaluations. We use $n \in \{10, 100, 1000, 10000, 100000\}$ and perform each measurement 100 times.

Evaluation Results. Figure 5.8 presents the mean running time of the decryption algorithm for each n and each variant (without and with re-encryptions). We can see that the decryption time without re-encryptions are mostly constant up to 1000 homomorphic evaluations, but increases drastically for larger numbers of evaluations. The reason for this sharp increase in decryption time is likely the fact that the discrete logarithms can no longer be computed via table lookup but the decryption has to fall back to exhaustive search (cf. Section 5.6.1). In comparison, when re-encryptions are performed, the decryption time only increases minimally, even for $n = 100000$.

5.7.3 Secure Neural Networks in the Cloud

In this experiment we consider the use case of evaluating neural networks in the cloud. We aim to protect the network model and the instance classified, i.e., the weights of the connections and the inputs and outputs of the neurons. The weights do not change between classifications and often represent intellectual property of the client. Also, the privacy of a user classified by the network is at risk, since the user's classification may be revealed. DFAuth overcomes these concerns by encrypting the weights in the network and the client's input and performing only encrypted calculations. Also, since the transformed program

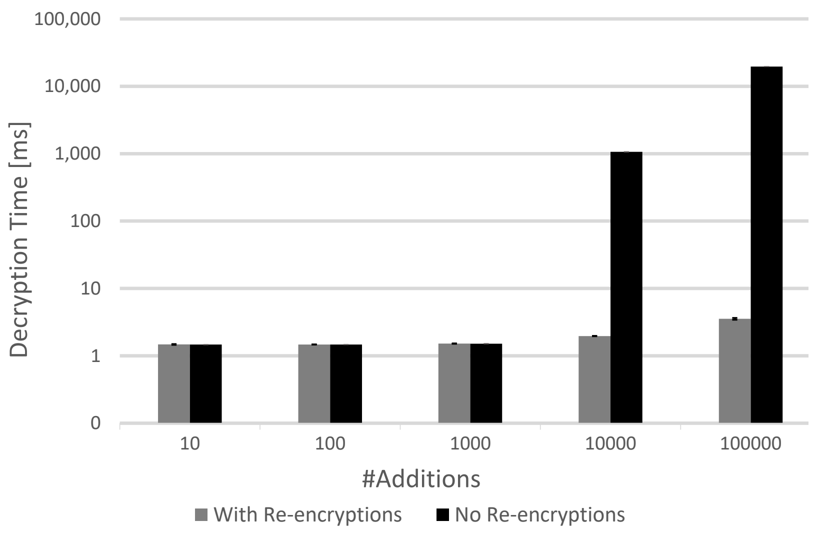


Figure 5.8: Mean running time [ms] to decrypt a ciphertext produced by summing up a varying number of ciphertexts using Additive HASE.

does not perform any control-flow decisions based on network weights or client input, the attacker cannot learn sensitive data by observing the control flow. Note that even the classification result does not leak, since the result returned is the output values for each of the classification neurons, i.e. a chance of classification y, e.g., breast cancer in our subsequent example, of $x\%$.

Experimental Setup. We apply our transformation to the BrestCancerSample [sic] neural network provided by Neuroph [Neu], a framework for neural networks. Given a set of features extracted from an image of a breast tumor, the network predicts whether the tumor is malignant or benign. As such, it operates on highly sensitive medical data.

The properties of the network, e.g., layer and neuron configuration, are encoded programmatically in the main class of the network. This class is also capable of reading the data set associated with the network and dividing it into a 70% training set and a 30% test set. We use the training set to learn network weights and the test set to evaluate whether the network delivers correct predictions.

First, we apply DFAuth to the main class of the network and the classes of the framework (app classes). Result of the transformation is a new main class and a set of app classes operating on ciphertexts rather than floating-point double values. For HASE, floating-point numbers are converted to integers by scaling by a factor of 10^6. Essentially, the resulting code allows us to evaluate DFAuth-encrypted neural networks in addition to plaintext neural networks.

Then, we test different network evaluation sizes $n \in \{1, 10, 20, \ldots, 100\}$ and perform 20 evaluation runs each. In each run, a new neural network is trained based on a random segmentation of training and test data. We use the facilities provided by Neuroph to serialize the trained network weights into a double array

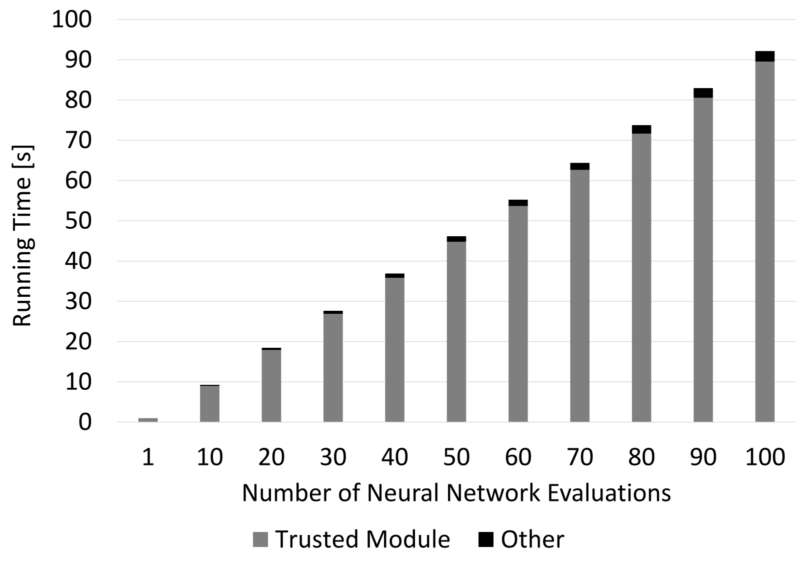


Figure 5.9: Mean running time [s] of the HASE variant of the breast cancer neural network experiment as a function of the number of evaluations.

and write the network to disk. Additionally, we encrypt the trained network weights using HASE and TACO and also write these encrypted neural networks to disk.

We end each run by performing n neural network evaluations with each of the three neural network variants stored on disk. Neural network inputs are sampled uniformly at random (without replacement) from the test data set. We measure the total running time of code executing at the untrusted server, the time spent invoking and inside the trusted module and the number of operations performed on encrypted data inside and outside of the trusted module. The total running time includes reading the network configuration (i.e., layers and neuron), loading the weights and executing the evaluation.

Evaluation Results. Figure 5.9 presents the mean running time of the encrypted neural network using DFAuth with HASE. The mean is computed over all 20 runs for each evaluation size and is divided into time spent inside the trusted module and time spent outside the trusted module. Figure 5.10 shows the results for DFAuth with TACO. Figure 5.11 compares the running times of HASE and TACO. We do not include the plaintext measurements in the graphs because they are too small to be visible. Table 5.3 reports the number of untrusted and trusted module (SGX) operations.

Using HASE, the total running time of one network evaluation is 951 ms, of which 895 ms (94.1%) are spent in the trusted module (SGX) and 56 ms (5.9%) outside of the trusted module. Even for 100 evaluations a run completes in 92.15 s on average. In this case, the processing time in the trusted module is 89.54 s (97.2%) and 2.60 s (2.8%) outside. The relative running time of an evalu-

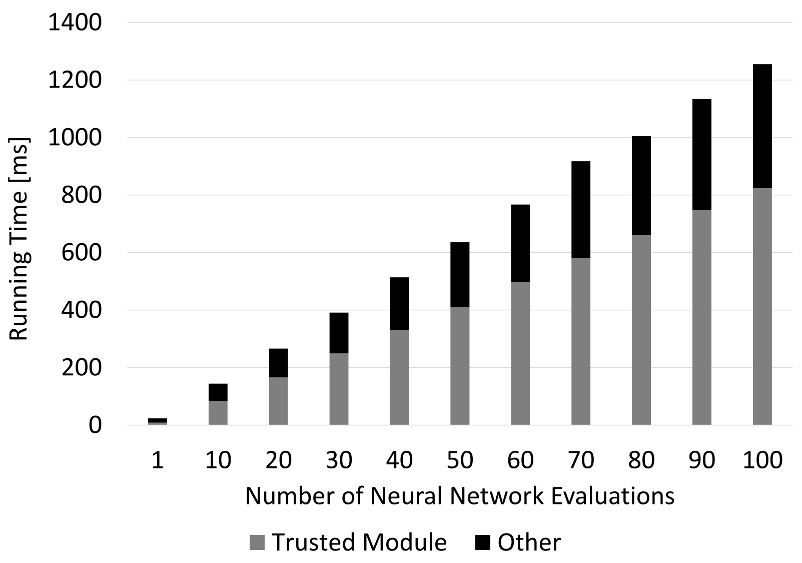


Figure 5.10: Mean running time [ms] of the TACO variant of the breast cancer neural network experiment as a function of the number of evaluations.

ation (total running time / number of network evaluations) is 921 ms. Compared to one plaintext network evaluation, the running time increased by a factor of about 1417. A waiting time of less than one second demonstrates the practical deployment of neural network evaluation on encrypted data and should already be acceptable for most use cases.

In Figure 5.9 we can see that using HASE a significant portion of the total running time is spent inside (or invoking) the trusted module. On the one hand, this shows that a more efficient trusted module implementation would significantly decrease the total running time of the application. On the other hand, it suggests that we execute more instructions inside the trusted module than outside, contradicting our basic idea of a reduced execution inside the trusted module.

Operation	HASE		TACO	
	Type	**# Ops**	**Type**	**# Ops**
Addition	HOM	548	SGX	548
Multiplication	HOM	548	SGX	548
Additive to Multiplicative Conversion	SGX	36		
Multiplicative to Additive Conversion	SGX	548		
Comparison	SGX	36	SGX	36
Total	HOM	1096	SGX	1132
	SGX	620		

Table 5.3: Number of untrusted homomorphic (HOM) and trusted (SGX) operations for a single evaluation of the neural network.

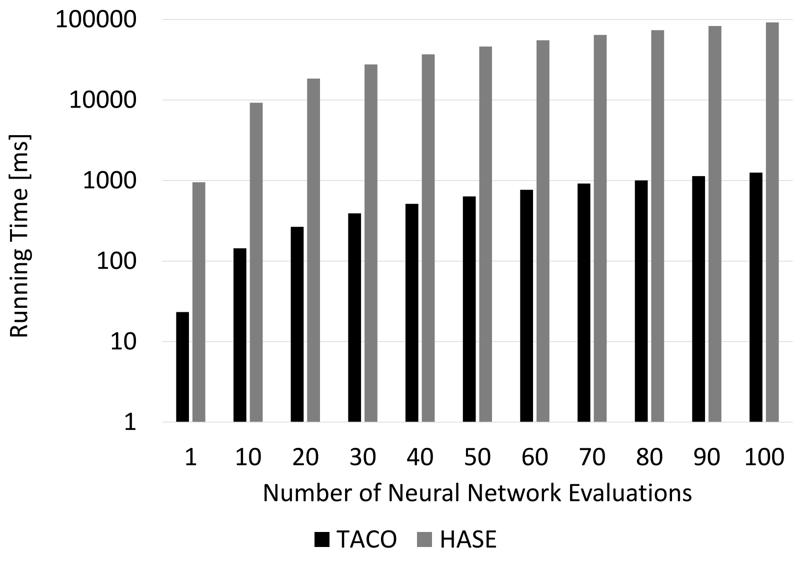


Figure 5.11: Compared mean running time [ms] of the TACO and the HASE variant of the breast cancer neural network experiment as a function of the number of evaluations.

However, Table 5.3 shows that this is not the case. For a single neural network evaluation, 1096 untrusted operations and 620 trusted operations on encrypted data are performed. This means that 64% of all operations can be performed without the trusted module.

Using TACO, the total running time of one network evaluation is 23 ms, of which 8 ms (35.7%) are spent in the trusted module (SGX) and 15 ms (64.3%) outside of the trusted module. For 100 evaluations the relative running time of an evaluation is 12.55 ms. In Figure 5.10 we can see that in contrast to HASE a smaller fraction of the total running time is spent in the trusted module. Although TACO requires 1132 trusted module invocations while HASE only requires 620, it appears that the higher number of invocations is easily compensated by TACO's use of a more efficient encryption scheme. For one network evaluation, TACO is faster than HASE by a factor of 41 and slower than plaintext evaluation by a factor of 35. See Figure 5.11 for a more detailed comparison of HASE and TACO running times.

Comparison to Alternative Solutions. Recently, implementations of machine learning on encrypted data have been presented for somewhat homomorphic encryption [Gil+16] and Intel SGX [Ohr+16]. Compared to the implementation on SWHE our approach offers the following advantages:

1. Our approach has a *latency* of 12.55 milliseconds compared to 570 seconds for SWHE. The construction by Gilad-Bachrach et al. [Gil+16] exploits the inherent parallelism of SWHE to achieve a high throughput. However, when evaluating only one sample on the neural network, the latency is high.

Our approach is capable of evaluating only a single sample with low latency as well.

2. Our approach scales to different machine learning techniques with *minimal developer effort*. Whereas the algorithms by Gilad-Bachrach et al. [Gil+16] were developed for a specific type of neural network, our implementation on encrypted data was derived from an existing implementation of neural networks on plaintext data by compilation. This also implies that the error introduced by Gilad-Bachrach et al. [Gil+16] due to computation on integers does not apply in our case. However, we have not evaluated this aspect of accuracy in comparison to their implementation.

3. Our approach is capable of *outsourcing* a neural network evaluation whereas the approach by Gilad-Bachrach et al. [Gil+16] is a two-party protocol, i.e., the weights of the neural network are known to the server. Our approach encrypts the weights of the neural network and hence a client can outsource the computation of neural network. Note that our approach includes the functionality of evaluating on plaintext weights as well and hence offers the larger functionality.

Although their running time overhead is smaller than ours, our approach offers the following advantage compared to the implementation on SGX [Ohr+16]: In our approach the code in the SGX enclave is *independent of the functionality*, e.g., machine learning. The implementation by Ohrimenko et al. [Ohr+16] provides a new, specific algorithm for each additional machine learning function, i.e., neural networks, decision trees, etc. Each implementation has been specifically optimized to avoid side-channels on SGX and hopefully scrutinized for software vulnerabilities. The same development effort has been applied once to our conversion routines and crypto library running in the trusted module. However, when adding a new functionality our approach only requires compiling the source program and not applying the same effort again on the new implementation.

5.7.4 Secure Electric Vehicle Charging Scheduling

In this experiment, we use DFAuth to protect sensitive data processed by a smart charging scheduler for EVs. Smart charging is a technique to schedule EV fleets making the most of existing infrastructure, for example undersized connection lines and a limited number of charging stations.

Frendo et al. [FGS19] present a novel approach combining day-ahead and real-time planning. Their schedule-guided heuristic takes as input a precomputed day-ahead schedule and adjusts it in real-time as new information arrives. Events providing new information for example include EV arrival, EV departure, and price changes at energy markets. Event processing must be fast, for example such that drivers can be assigned a charging station as they enter a parking garage. The event handling process is divided into two parts, EV scheduling and EV prioritization, as depicted in Figure 5.12.

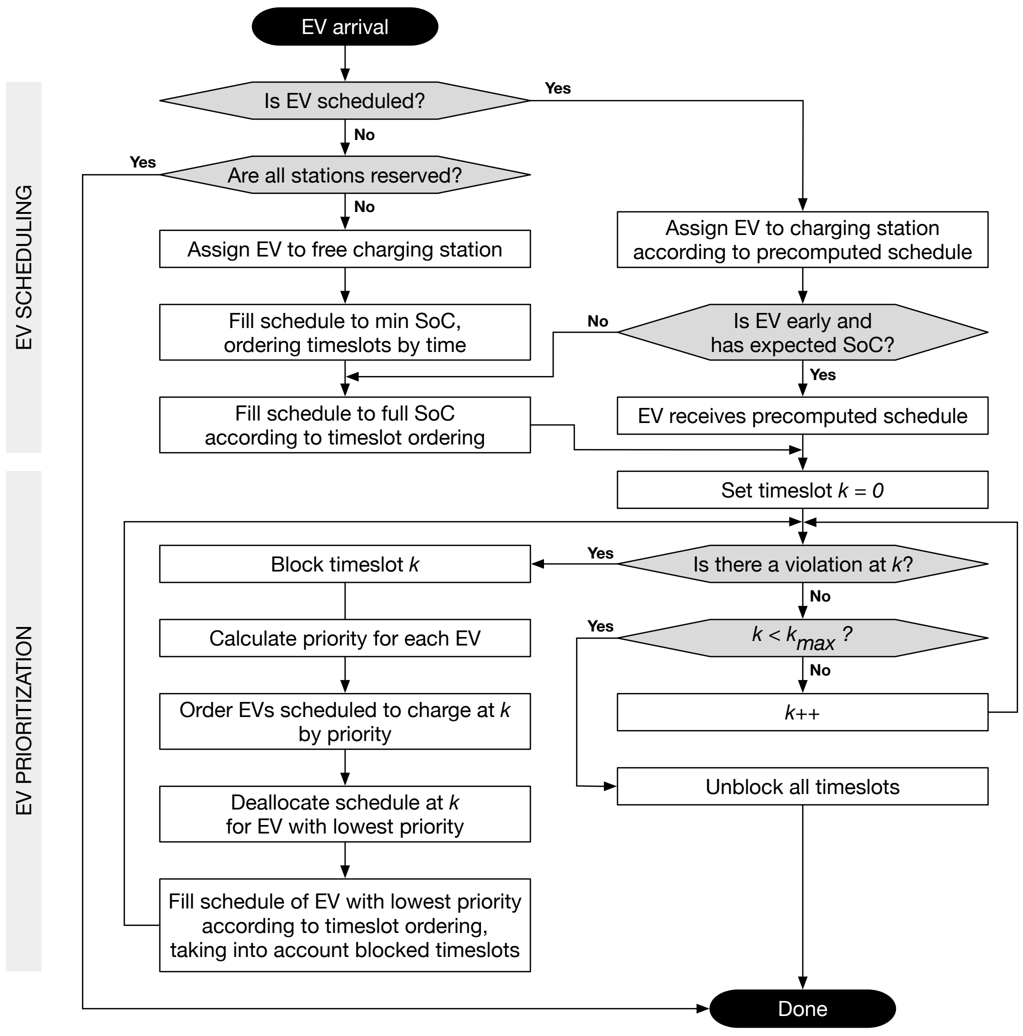

EV arrival
Is EV scheduled?
Yes
No
Are all stations reserved?
Yes
No
Assign EV to free charging station
Fill schedule to min SoC, ordering timeslots by time
Fill schedule to full SoC according to timeslot ordering
Assign EV to charging station according to precomputed schedule
Is EV early and has expected SoC?
No
Yes
EV receives precomputed schedule
Set timeslot $k = 0$
Is there a violation at k?
Yes
No
Block timeslot k
Calculate priority for each EV
Order EVs scheduled to charge at k by priority
Deallocate schedule at k for EV with lowest priority
Fill schedule of EV with lowest priority according to timeslot ordering, taking into account blocked timeslots
$k < k_{max}$?
Yes
No
k++
Unblock all timeslots
Done
EV SCHEDULING
EV PRIORITIZATION

Figure 5.12: Schedule-guided smart charging heuristic (taken from Frendo et al. [FGS19]).

To provide event responses in real-time, it is convenient to utilize the cloud's powerful computing resources, high availability and central data storage. However, sensitive data such as planned EV arrival and departure times as well as technical car properties, which could be used to identify the driver, are at risk of being revealed to the cloud service provider. We apply DFAuth such that all operations involving sensitive data are performed on encrypted data.

Experimental Setup. Overall, we apply DFAuth to the schedule-guided heuristic by Frendo et al. [FGS19] and reproduce their real-time charging simulation using our *encrypted schedule-guided heuristic*. Provided a precomputed *encrypted day-ahead schedule* as input, we simulate one day of events during which the current schedule has to be updated.

In particular, we apply our transformation to all classes handling sensitive data. For each car, the information protected in this way includes its minimum state of charge (min SoC), its current state of charge and its current charging schedule. We do not protect prices at energy markets, the structure of the charging schedule (e.g., timeslot size) and the car type, i.e., whether it is a battery-electric or a plug-in hybrid electric vehicle. Result of the transformation are new classes operating on ciphertexts rather than floating-point double values. For HASE, floating-point numbers are converted to fixed-point numbers by scaling by a factor of 10^8.

The charging simulation is parameterized by the number of EVs. We consider the same parameters $\{10, 20, 30, \ldots, 400\}$ as in the original experiment [FGS19]. The number of charging stations is fixed at 25. Possible events are EV arrival, EV departure, update of expected EV arrival, update of expected EV departure and price changes at energy markets. Each EV event occurs approximately once per EV, new energy prices are updated every 15 minutes, i.e., 96 times per simulation.

We execute the simulation for each parameter for HASE and TACO using the same underlying data set. In each simulation, we measure the accumulated time taken to adjust the schedule as a result of an event. We distinguish the running time of code executing at the untrusted server, the time spent invoking and inside the trusted module and the number of operations performed on encrypted data inside and outside of the trusted module. To evaluate the real-time property of our solution, we also determine the maximum event response time over all events.

Evaluation Results. Figure 5.13 presents the total running time of each simulation for HASE and TACO. It can be seen that TACO also outperforms HASE in this experiment. The smallest simulation using 10 EVs takes 802 s for HASE, but only 4 s for TACO. Even for the largest simulation involving 400 EVs, the running time of 139 s for TACO is significantly lower than for 10 EVs using HASE. The graph does not include any HASE results for parameters larger than 20, because we aborted the HASE experiment due to its long running time. In Figure 5.13 we can also see that sometimes the running time of a simulation decreases although the number of EVs increases, e.g., for parameters 220 and 230. This is caused by randomized sampling of simulation data, which may result in larger instances being easier to solve by the heuristic than smaller instances.

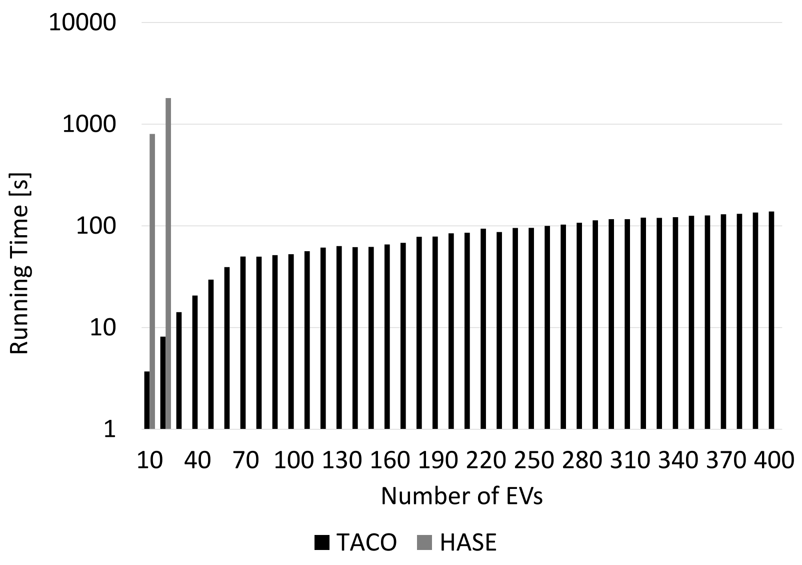


Figure 5.13: Running times of EV simulations.

Figure 5.14 shows the number of trusted module calls for each simulation. The trusted module calls linearly increase in the number of EVs, as is expected based on the algorithm. For parameters 220 and 230, the number of trusted module calls are in line with the respective running times and again suggest that the simulation involving 230 EVs was easier to solve. As in the neural network experiment, HASE requires less trusted module calls than TACO. For example, for 10 EVs HASE requires 284 920 trusted module calls while TACO performs 309 656. This may result in better overall HASE performance in case an alternative trusted module with more expensive calls is used.

Figure 5.15 illustrates the time TACO spent inside and outside the trusted module. The portion of time inside the trusted module only varies between 59.5% for 320 EVs and 62.4% for 10 EVs, and as such can be considered independent of the number of EVs. In comparison, for HASE the average portion of time inside the trusted module is 99.5%, despite the lower number of trusted module calls and higher total running time.

Figure 5.16 presents the maximum event response time, an important property of real-time applications. By definition, the maximum response time is dominated by single events which are complex to handle for the heuristic. Such events cause spikes in the chart, as can be observed for 70 and 130 EVs. If we ignore those outliers, we can split the graph into two parts. From 10 to 90 EVs, the maximum response time continuously increases with the number of EVs. This is because time is mostly spent in the scheduling step and only a small number of violations have to be resolved in the prioritization step (cf. Figure 5.12). For larger numbers of EVs the maximum response time varies between 0.70 s and 2.14 s with an average of 1.06 s. Considering the response time of different simulations, we can see that it is practically independent of the number of EVs.

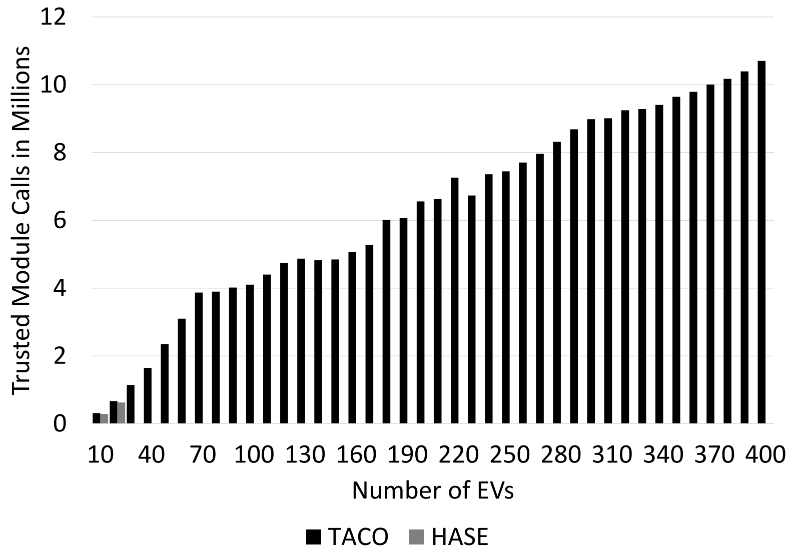


Figure 5.14: Number of trusted module calls in EV simulations.

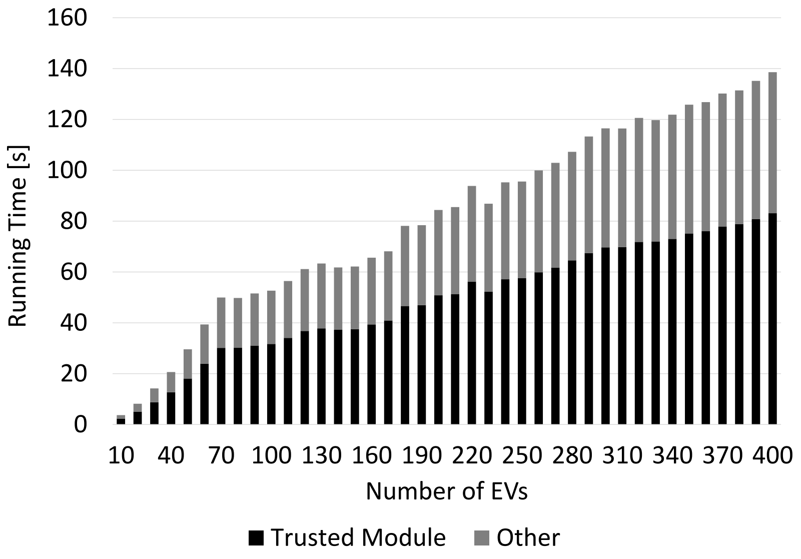


Figure 5.15: Running time inside and outside the trusted module for TACO in EV simulations.

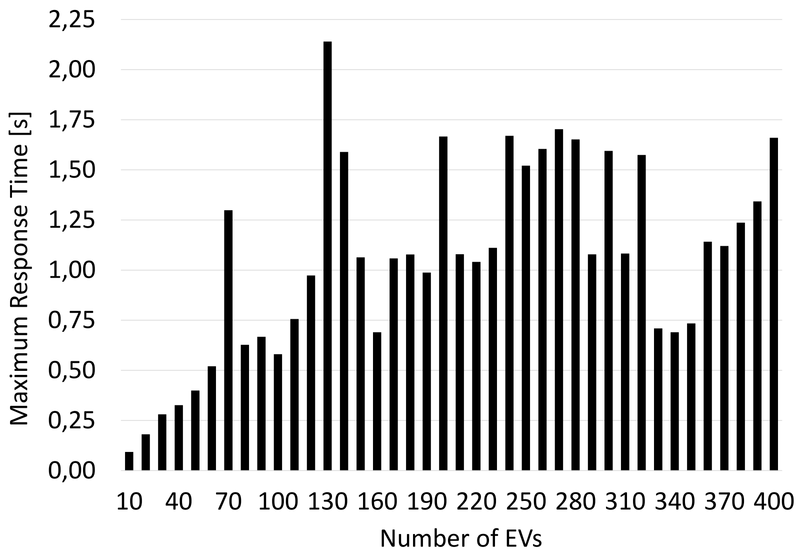


Figure 5.16: Maximum event response time for TACO in EV simulations.

In summary, DFAuth using TACO enables the protection of sensitive data while at the same time providing sufficiently fast response times required by the smart charging use case.

5.8 Related Work

The work presented in this chapter is related to obfuscation techniques and trusted hardware, (homomorphic) authenticated encryption and computation on encrypted data – including but not limited to homomorphic encryption. An overview on computation on encrypted data and related concepts is provided in Chapter 3.

Obfuscation Techniques and Trusted Hardware Approaches straightening or obfuscating the control flow can be combined with DFAuth on the unprotected program part and are hence orthogonal to DFAuth. We provide a detailed analysis of the security and performance implications of executing control-flow driven programs in Chapter 6. An introduction to trusted execution environments and a detailed description of Intel SGX is provided in Section 3.4.

Molnar et al. [Mol+05] eliminate control-flow side-channels by transforming code containing conditional instructions into straight-line code employing masking.

GhostRider [Liu+15] enables privacy-preserving computation in the cloud assuming a remote trusted processor. It defends against memory side-channels by obfuscating programs such that their memory access pattern is independent of control-flow instructions. However, as a hardware-software co-design, GhostRider requires a special co-processor. In contrast, DFAuth works on commodity SGXenabled CPUs and provides a program-independent TCB inside the secure hardware. Raccoon [RLT15] extends these protections to the broader class of side-channels carrying information over discrete bits. Essentially, Raccoon executes both paths of a conditional branch and later combines the real and the decoy path using an oblivious store operation.

HOP [Nay+17] obfuscates programs by encrypting them such that only a trusted processor can decrypt and run them. By incorporating encryption routines into the program, HOP can be extended to also protect program input and output. However, HOP assumes the program is free of software vulnerabilities and runs the entire program inside the trusted hardware. In contrast, in DFAuth vulnerabilities are confined to the untrusted program and the code inside the trusted module is program-independent.

(Homomorphic) Authenticated Encryption Authenticated encryption (AE) provides confidentiality as well as authenticity (cf. Section 2.2.3) and is the recommended security notion for symmetric encryption. An AE scheme can be obtained by composing an IND-CPA secure encryption scheme with a message authentication code (MAC) or signature scheme [BN08]. Hence, one can obtain a *homomorphic* AE scheme by combining a homomorphic encryption scheme (cf. Section 3.1.4) with a homomorphic MAC. However, since the best known homomorphic MACs [GW13] are not yet fully homomorphic, a different construction is required. Joo and Yun provide the first fully homomorphic AE [JY14]. However, their decryption algorithm is as complex as the evaluation on ciphertexts, undermining the advantages of an encrypted program, i.e., one could do the entire computation in the trusted module. In parallel work, Barbosa et al. [BCF17] develop labeled homomorphic encryption which, however, has not been applied to trusted modules.

Boneh et al. [Bon+09] introduced linearly homomorphic signatures and MACs to support the efficiency gain by network coding. However, their signatures were still deterministic, hence not achieving IND-CPA security. Catalano et al. [CMP14] integrated MACs into efficient, linearly homomorphic Paillier encryption [Pai99a] and used identifiers to support public verifiability, i.e., verification without knowledge of the plaintext. However, their scheme also has linear verification time undermining the advantages of a small trusted module. In our HASE construction we aimed for using identifiers and not plaintext values to enable DFAuth. Furthermore, we split verification into a pre-computed derivation phase and a verification phase. Hence, we can achieve constant time verification.

Aggregate MACs [KL08] provide support for aggregation of MACs from distinct keys. However, our current DFAuth considers one client and secret key.

Computation on Encrypted Data A detailed description of techniques transforming programs to compute on encrypted data is given in Section 3.3. The underlying efficient encryption schemes are introduced in Section 3.1. MrCrypt [Tet+13] infers feasible encryption schemes using type inference. In addition to homomorphic encryption, MrCrypt makes use of randomized and deterministic order-preserving encryption (cf. Section 3.1.1). However, the set of feasible programs is limited. JCrypt [DMD16] improved the type inference algorithm to a larger set of programs. However, neither MrCrypt nor JCrypt consider conversions between encryption schemes. AutoCrypt [Top+13] used these conversions, however, realized their security implications. The authors hence disallowed any conversion from homomorphic encryption to searchable encryption. This restriction prevents any program from running that modifies its input and then performs a control-flow decision. Such programs include the arithmetic computations we performed in our evaluation.

Verifiable computation [GGP11] can be used by a client to check whether a server performed a computation as intended – even on encrypted data. However, this does not prevent the attacks by malicious adversaries considered in this chapter. It only proves that the server performed one correct computation, but not that it did not perform any others.

Functional encryption (cf. Section 3.1.3) is a more powerful computation on encrypted data than homomorphic encryption. It not only can compute any function, but also reveal the result of the computation and not only its ciphertext. However, generic constructions are even slower than homomorphic encryption. Searchable encryption (cf. Section 3.1.2) is a special case of functional encryption for comparisons and could be used to implement comparisons in DFAuth. However, since the actual comparison time is so insignificant compared to the cryptographic operations, it is more efficient to implement comparison in the trusted module as well.

5.9 Summary

In this chapter, we introduced the concept of dataflow authentication (DFAuth) which prevents an active adversary from deviating from the dataflow in an outsourced program. This in turn allows safe use of re-encryptions between homomorphic and leaking encryption schemes in order to allow a larger class of programs to run on encrypted data where only the executed control flow is leaked to the adversary. Our implementation of DFAuth uses two novel schemes, homomorphic authenticated symmetric encryption (HASE) and trusted authenticated ciphertext operations (TACO), and trusted modules in an SGX enclave. Compared to an implementation solely on fully homomorphic encryption we offer better and actually practical performance and compared to an implementation solely on Intel's SGX we offer a much smaller trusted code base independent of the protected application. We underpin these results by an implementation of a bytecode-to-bytecode compiler that translates Java programs into Java programs operating on encrypted data using DFAuth.

6 PASAPTO: Policy-Aware Security and Performance Trade-off Analysis

In this chapter, we address the concern that control-flow leakage might be inacceptable from a security perspective. To avoid unconditionally sacrificing performance for security, we analyze the trade-off between the two properties in more detail. To this end, we extend dataflow authentication with a policy-aware security and performance trade-off (PASAPTO) analysis.

Section 6.1 describes our motivation in more detail and presents an illustrative example program. Section 6.2 introduces our program and adversary models as well as our definitions of policy-compliance and trade-off analysis. Section 6.3 explains how to quantify adversarial information flow resulting from control-flow observations. Section 6.4 presents the PASAPTO optimization problem and two heuristics approximating a solution. Section 6.5 constructs a PASAPTO analysis for DFAuth. Section 6.6 provides details about the implementation of our DFAuth trade-off analyzer (DFATA). Section 6.7 presents the results of our experiments using this implementation. Section 6.8 discusses related work and Section 6.9 provides a summary of this chapter.

The content of this chapter has been the subject of the following scientific publications, of which parts are included verbatim in this thesis.

- *Andreas Fischer, Jonas Janneck, Jörn Kussmaul, Nikolas Krätzschmar, Florian Kerschbaum, Eric Bodden: PASAPTO: Policy-aware Security and Performance Trade-off Analysis – Computation on Encrypted Data with Restricted Leakage. In 33rd IEEE Computer Security Foundations Symposium, 2020.* [Fis+20b]

6.1 Introduction

Efficient computation on encrypted data without side-channels remains an open problem. On the one hand, cryptographic techniques such as FHE (cf. Section 3.1.4) can perform arbitrary computations without revealing any information about the data computed on, but suffer high computational costs. On the other hand, TEEs such as Intel SGX enclaves (cf. Section 3.4.1) entail only little computational overhead, but are vulnerable to side-channel attacks (cf. Section 3.4.2). For example, it has been demonstrated that the control flow executed inside SGX enclaves can be inferred from untrusted programs [Lee+17b]. Similarly, DFAuth (cf. Chapter 5) considers any control-flow decision on an encrypted variable an intentional leak by the programmer.

```
int p1(int x)
    if (e)
        y = f(x)
    else
        y = g(x)
    return y
```

```
int p2(int x)
    y1 = f(x)
    y2 = g(x)
    # obliviously select
    # y1 or y2 based on e
    y = select(e, y1, y2)
    return y
```

Figure 6.1: Two semantically equivalent variants of a program.

For some applications, meaningful security can only be achieved when all side-channels are eliminated. Consider, for example, cryptographic primitives such as square-and-multiply algorithms used for modular exponentiation in public key cryptography. If private key bits are leaked through side-channels [BB03], all security relying on the secrecy of the private key is lost. A broad class of side-channels can be avoided by producing constant-time code not making any memory accesses or control-flow decisions based on secret data. Some cryptographic primitives have even been designed such that an implementation likely possesses these properties [Ber06]. However, for more complex applications these properties cannot be achieved without causing prohibitive performance. Consider, for example, Dantzig's simplex algorithm [Dan51] for solving linear optimization problems. This algorithm terminates as soon as objective values of the current solution can no longer be improved. It is extraordinarily efficient in practice but its worst-case running time is exponential in the problem size [KM70]. If we are to eliminate all side-channels, we must also prevent the termination condition from leaking. Thus, any simplex invocation must have exponential running time. Since this behavior is impractical for any non-trivial input, engaging in a trade-off between security and performance seems justified.

In this work, we consider the trade-off between security and performance of control-flow side-channels when executing programs on encrypted data. Our motivation for focusing on control flow is based on the expectation that the disclosure of control-flow information, e.g., rather than computing a circuit as in FHE, results in significant performance gains.

Consider, for example, the two variants of a program provided in Figure 6.1. The first program `p1` reveals the control flow – more precisely the boolean result of the conditional expression e – to an attacker capable of observing the executed control flow. Based on the result of e, only either f or g is computed. The second program `p2` computes both f and g and combines the result by invoking an oblivious `select` function. Because no control-flow decisions can be observed, the attacker does not learn the result of e. Since `p2` has to perform $c(f) + c(g)$ computations while `p1` has to perform only $\max\{c(f), c(g)\}$ computations, we expect `p2`'s performance to be inferior to that of `p1`.

More generally, a trade-off for a program is determined by selecting for each control-flow decision whether it may be *revealed* or must be *hidden*. Since the number of program variants is exponential in the number of control-flow decisions and not all selections have the same impact on security and performance, it is impracticable for developers to make such selections manually for any non-trivial program. Our analysis uses program transformation techniques to hide control-flow decisions and explore the security-performance trade-off.

To enable strong security guarantees, our analysis incorporates information-flow policies, which allow developers to define varying sensitivity levels on data. By defining the appropriate sensitivity level, developers can also completely prevent data from being revealed during program execution.

We first formalize the problem of policy-aware security and performance trade-off (PASAPTO) analysis as an optimization problem and prove the NP-hardness of the corresponding decision problem. Then, to make the problem tractable, we develop two heuristic algorithms computing an approximation of the Pareto front of the optimization problem. Finally, we extend DFAuth with a PASAPTO analysis to investigate the trade-off of control-flow leakage. Our DFAuth trade-off analyzer (DFATA) takes as input a program operating on plaintext data and an associated information flow policy. It outputs semantically equivalent program variants operating on encrypted data which are policy-compliant and approximately Pareto-optimal with respect to security and performance.

DFAuth is not the only application of our PASAPTO analysis, but the same analysis can be applied to investigate the security-performance trade-off of other techniques revealing partial information about secret data through control-flow. For example, the same analysis can be applied to control-flow driven programs running in SGX enclaves (without the help of DFAuth). Also, this trade-off does not only occur when outsourcing sensitive computations, but the same trade-off can be made in many other types of computation on encrypted data. For example, it also applies to MPC protocols (cf. Section 3.2.1). By declassifying (i.e., making public) partial information such as intermediate results, expensive MPC computations can be avoided and the performance of MPC protocols can be improved.

6.2 Definitions

To better understand our analysis, we first introduce our program and adversary models as well as our definitions of policy-compliance and trade-off analysis.

6.2.1 Programs and Computation

We model a program as a transition system $P = (S, Tr, I, F)$ consisting of a set of *possible program states* S, a set of program *transitions* Tr, a set of *initial states* $I \subseteq S$ and a set of *final states* $F \subseteq S$. We use $\mathcal{P}$ to refer to the *set of all programs.*

A program state $s \in S$ contains an *instruction pointer* and a map assigning each program variable a value from its domain. We refer to the *set of variables* of a program as V. Without loss of generality, we assume all variables to be integers in the range $D_n := [-2^{n-1}, 2^{n-1} - 1] \subset \mathbb{Z}$.

Each transition corresponds to a program *statement* as written in a programming language. We consider a deterministic, imperative programming language with assignments, conditional expressions, et cetera. We refer to a transition corresponding to a program statement containing a control-flow decision as a *control-flow transition.* By $T \subseteq Tr$ we denote the set of all control-flow transitions.

6.2.2 Adversary Model

We consider a passive adversary capable of continuously observing the instruction pointer of the program state $s \in S$ during an execution of a program. We assume the adversary knows the program text, that is, any transition and e.g., any program constant. Hence, the adversary is capable of continuously observing the control flow. The goal of the adversary is to learn additional information about the initial state $i \in I$, i.e., which input variable is assigned which value.

6.2.3 Information Flow Policy Compliance

An attacker may learn information about the program state from expressions determining control-flow decisions. To prevent sensitive data from being revealed, it is thus desirable to ensure that such data is either not used in a control-flow decision or the adversarial information flow caused by the control-flow decision does not exceed a certain user-defined threshold.

In this section, we define *information flow policies* allowing these requirements to be expressed on a per-variable basis. Our goal is to verify that a given program P complies with a given information flow policy. If P is non-compliant, our goal is to use a *control-flow removal algorithm* to transform P into a semantically equivalent program P' that is compliant.

Definition 46 (Quantitative Information Flow Policy)**.** *Let P be a program and V the set of variables in P. A* quantitative information flow policy *Ψ is a map assigning each program variable a numeric upper bound on the adversarial information flow when executing P. Formally,*

$$\Psi : V \to \mathbb{R}_{\geq 0} \cup \{\infty\}$$

where we use ∞ to denote no upper bound on the adversarial information flow.

Definition 47 (Policy Compliance)**.** *Let P be a program with variables V and let Ψ a quantitative information flow policy for P. We say that P* is compliant with *Ψ iff for each variable $v \in V$ the adversarial information flow for v when executing P does not exceed $\Psi(v)$. If P is not compliant with Ψ, we say that P* violates *Ψ.*

Definition 48 (Control-Flow Removal Algorithm)**.** *A* control-flow removal algorithm

$$\mathcal{T} : \mathcal{P} \times \{0,1\}^{|T|} \to \mathcal{P}$$

takes as input a program P and a binary vector t specifying for each control-flow transition $\tau_i \in T$ whether it may be revealed *($t_i = 1$) or must be* hidden *($t_i = 0$).*

It outputs a semantically equivalent program variant P' only containing control-flow transitions τ_i where $t_i = 1$. We require that $\mathcal{T}$ does not introduce any new control-flow transitions.

6.2.4 Security-Performance Trade-off Analysis

Two fundamental requirements for an analysis of the trade-off between security and performance of a policy-compliant program are a function measuring the security of a given program and a function measuring the performance of a program. We model both as cost functions, i.e., in a *the lower the better* fashion. Finally, with all required components introduced, we are ready to define the key problem considered in this chapter.

Definition 49 (Program Security Measure)**.** *A* program security measure *is a function assigning a non-negative numeric value to a given program.*

$$\mu_s : \mathcal{P} \to \mathbb{R}_{\geq 0}$$

Definition 50 (Program Performance Measure)**.** *A* program performance measure *is a function assigning a non-negative numeric value to a given program.*

$$\mu_p : \mathcal{P} \to \mathbb{R}_{\geq 0}$$

Definition 51 (Policy-Aware Security-Performance Trade-off Analysis)**.** *Let P be a program and Ψ a quantitative information flow policy for P. Furthermore, let μ_s be a program security measure, μ_p a program performance measure and $\mathcal{T}$ a control-flow removal algorithm. The problem of* policy-aware security and performance trade-off (PASAPTO) analysis *is to produce – in expected polynomial time – a set of programs $\mathbb{P}$ such that each program $P' \in \mathbb{P}$ is*

- *a $\mathcal{T}$-transformation of P, that is $P' = \mathcal{T}(P, t)$ for some binary vector $t \in \{0,1\}^{|T|}$,*
- *compliant with policy Ψ and*
- *Pareto-optimal with respect to (μ_s, μ_p).*

6.3 Control Flow Leakage Quantification

In this section, we describe how adversarial information flow (leakage) resulting from the observation of control-flow decisions can be quantified. Section 6.3.1 introduces two program security measures (instances of μ_s). Section 6.3.2 shows how to capture leakage for each variable of a program such that we can support quantitative information flow policies.

6.3.1 Two Program Security Measures

To evaluate the security of programs, we rely on information theory (cf. Section 2.3) and other established quantitative information flow (QIF) techniques [BKR09; Kle14; Mal11; Smi09]. Our QIF analysis is decomposed into two steps: an *algebraic interpretation* followed by a *numerical evaluation*. We first capture the view of any adversary as an equivalence relation on the initial states of a program. Then, we quantize the equivalence relation to arrive at a numeric value expressing the adversarial information flow when executing the program. The primary benefit of a two-step QIF analysis is that each of the two steps can be considered independently. For example, multiple numeric evaluations can be defined based on the same algebraic interpretation.

Algebraic Interpretation We model the information flow to an observer resulting from an execution of a program as an equivalence relation R on its initial states. $R \subseteq I \times I$ relates two states if an observer cannot distinguish between them. R is called *indistinguishability relation* and induces an *indistinguishability partition* $I_{/R}$ on the set of all possible initial states I.

In the extreme case of $R = I \times I$, the observer cannot distinguish between any of the program's states and has learned nothing from its observation. If, on the other hand, the equivalence classes are singleton sets, the observer has perfect knowledge of the program's initial state. Intuitively, the higher the number of equivalence classes and the smaller the classes, the more information is revealed to the attacker.

The indistinguishability partition can be computed using symbolic execution [Kin76], a method of program evaluation using symbolic variables and expressions rather than actual input values. More precisely, symbolic execution can be used to obtain symbolic program paths which aggregate input values leading to the same control flow. The resulting equivalence classes contain input values indistinguishable from the point of view of an adversary. Overall, we assume the existence of an algorithm Π performing symbolic execution to map a program $P = (S, Tr, I, F)$ to its indistinguishability partition, i.e., $\Pi(P) := I/_R$.

Numerical Evaluation For the second step, a multitude of valuations have been proposed in the literature [BKR09; Kle14; Mal11]. It should be noted that neither of the established definitions is superior to any other definition in all cases. The valuation to use depends on the system model, the adversary model and the question that is supposed to be answered. We are concerned with the question "How much information can an attacker gain from observing the system?" [Mal11] and define two security metrics measuring the answer in bits.

We measure the uncertainty of an attacker about the initial state of a program using the information-theoretic concept of Shannon entropy (Definitions 19-21). The adversarial information flow (leakage) is then defined as the reduction in uncertainty before and after observing the program's control flow. In the following, we first introduce two random variables, then we define two information flow measures. The *average information flow* measures the leakage of a program using weighted averaging over all equivalence classes. The *maximum information flow* only considers the smallest equivalence class, thus it measures the worst-case leakage of a program.

Let $P = (S, Tr, I, F)$ be a program and R its indistinguishability relation after observing the control flow. Define two discrete random variables [BKR09]:

- $\mathcal{U} : I \to I$ with probability distribution $p : I \to \mathbb{R}$ represents the distribution of the initial states.
- $\mathcal{V}_R : I \to I/_R$ maps each initial state to its equivalence class according to R.

Definition 52 (Average Information Flow). *Let P, $\mathcal{U}$, $\mathcal{V}_R$ be as before and let H denote Shannon entropy. We define the* average information flow ($\bar{\mu}$) *of P as*

$$\bar{\mu}(P) := H(\mathcal{U}) - H(\mathcal{U}|\mathcal{V}_R).$$

Definition 53 (Maximum Information Flow). *Let P, $\mathcal{U}$, $\mathcal{V}_R$ be as before and let H denote Shannon entropy. We define the* maximum information flow (μ_{max}) *of P as*

$$\mu_{max}(P) := H(\mathcal{U}) - H_\infty(\mathcal{U}|\mathcal{V}_R).$$

Example We illustrate our adversary model and the definitions of indistinguishability partition and information flow measures using the following program P.

```
1  int example1(int x)
2      if (x > 42)
3          x *= 2
4      return x
```

Any possible state of P contains one variable x. To simplify the exposition, let us assume that the domain of x is $D_8 = [-128, 127]$, i.e. x is a signed 8-bit integer. As the adversary knows the program text, the domain of x is known before the execution of the program. Hence the indistinguishability partition contains a single equivalence class.

$$I/_{R_{start}} = \{\{x \in D_8\}\}$$

By observing the control flow – more precisely whether the then-branch (line 3) is taken or not – the attacker can infer whether or not $x > 42$. The indistinguishability partition of the program after observing the control flow (after program execution) is thus

$$I/_R = \{\{x \in D_8 \mid x > 42\}, \{x \in D_8 \mid x \leq 42\}\}.$$

If we assume $\mathcal{U}$ to be distributed uniformly, i.e. each initial state is equally likely, then the initial uncertainty is

$$H(\mathcal{U}) = -\log \frac{1}{2^8} = 8\,\text{bits}.$$

The uncertainty after the control-flow observation is given by the conditional entropy. Each equivalence class $E \in I/_R$ occurs with probability $\frac{|E|}{|I|}$, thus we obtain

$$\begin{aligned} H(\mathcal{U}|\mathcal{V}_R) &= -\sum_{E \in I/_R} \frac{|E|}{|I|} \log |E| \\ &= -\frac{85}{256} \log \frac{1}{85} - \frac{171}{256} \log \frac{1}{171} \qquad (6.1) \\ &= 7.08\,\text{bits}. \qquad (6.2) \end{aligned}$$

The average information flow of program P is

$$\begin{aligned} \bar{\mu}(P) &= H(\mathcal{U}) - H(\mathcal{U}|\mathcal{V}_R) \\ &= 8 - 7.08 \\ &= 0.92\,\text{bits}. \end{aligned}$$

In (6.1) we can see that the then-branch is less likely than the (empty) else-branch but leads to a higher amount of information flow. To determine the worst-case leakage we compute the maximum information flow

$$\begin{aligned} \mu_{max}(P) &= H(\mathcal{U}) - H_\infty(\mathcal{U}|\mathcal{V}_R) \\ &= 8 - \left(-\log \frac{1}{85}\right) \\ &= 1.59\,\text{bits}, \end{aligned}$$

which only considers the equivalence class associated with the branch resulting in the largest adversarial information flow.

6.3.2 Variable-Specific Information Flow

As defined by Definition 47, we consider a program P with variables V compliant with a quantitative information flow policy Ψ if the adversarial information flow of P does not exceed $\Psi(v)$ for any variable $v \in V$. We can express this requirement formally by considering an information flow measure $\mu_{var}(P, v)$ which provides the adversarial information flow for a specific variable v of a program P. Naturally, policy-compliance can then be defined as

$$P \text{ is compliant with } \Psi \Leftrightarrow \forall v \in V : \mu_{var}(P, v) \leq \Psi(v).$$

Since $\Psi(v)$ defines an *upper bound* on the adversarial information flow for variable v, the measure $\mu_{var}(P, v)$ must be a worst-case information flow measure. In the remainder of this section, we hence construct a $\mu_{var}(P, v)$ measure based on the maximum information flow program security measure (Definition 53) introduced in the previous section.

Consider random variable $\mathcal{U}$ which distinguishes the initial states from each other. To define a variable-specific measure for variable v_i, we want to distinguish only states in which values of v_i differ. To this end, we consider a partition on the initial states fulfilling the following relation. Two states are equivalent if and only if the variable assignments of the i-th variable are equal. We denote this partition as $I_{/v_i}$ and define the random variable $\mathcal{U}_{|v_i} : I \to I_{/v_i}$. We can now define a variable-specific information flow measure.

Definition 54 (Variable-Specific Information Flow)**.** *Let P be a program with variables V. Furthermore, let H, $\mathcal{U}$, $\mathcal{V}_R$ be as in Section 6.3.1. We define the* variable-specific information flow *of a variable $v_i \in V$ as*

$$\mu_{var}(P, v_i) := H(\mathcal{U}_{|v_i}) - H_\infty(\mathcal{U}_{|v_i}|\mathcal{V}_R).$$

Example Consider the following program P given by its program code.

```
int example2(int x1, int x2)
    if (x1 > 42)
        x1 += 2
    if (x2 == 42)
        x1 *= 2
    return x1
```

Now consider the information flow policy Ψ defined by $\Psi(x_1) := 2$ and $\Psi(x_2) := \infty$. To simplify the exposition, we assume the domain of x_1 and x_2 to be $D_8 = [-128, 127]$ and the inputs to be distributed uniformly. The initial state then is $I = \{(x_1, x_2) \in D_8 \times D_8\}$ and the attacker's initial uncertainty is $H(\mathcal{U}) = \log 2^{8+8} = 16\,\text{bits}$. Using symbolic execution algorithm Π we can obtain the indistinguishability partition as

$$\begin{aligned} I/_R = \{&\{(x_1, x_2) \in D_8 \times D_8 \mid x_1 > 42 \wedge x_2 = 42\}, \\ &\{(x_1, x_2) \in D_8 \times D_8 \mid x_1 > 42 \wedge x_2 \neq 42\}, \\ &\{(x_1, x_2) \in D_8 \times D_8 \mid x_1 \leq 42 \wedge x_2 = 42\}, \\ &\{(x_1, x_2) \in D_8 \times D_8 \mid x_1 \leq 42 \wedge x_2 \neq 42\}\}. \end{aligned}$$

The worst-case overall uncertainty after observation of control flow amounts to $H_\infty(\mathcal{U}|\mathcal{V}_R) = 6.41\,\text{bits}$, thus the maximum information flow of the program is $\mu_{max}(P) = 16 - 6.41 = 9.59\,\text{bits}$.

To check compliance to Ψ, we have to investigate the information flow of each individual variable. For variable x_2 compliance is trivially fulfilled. Performing a projection on variable x_1 leads to an initial uncertainty of

$$H(\mathcal{U}_{|x_1}) = \log 2^8 = 8\,\text{bits}$$

and an uncertainty after observation of

$$H_\infty(U_{|x_1}|V_R) = \log 85 = 6.41\,\text{bits}.$$

We obtain

$$\mu_{var}(P, x_1) = 8 - 6.41 = 1.59\,\text{bits}.$$

Since $\mu_{var}(P, x_1) \leq \Psi(x_1)$ and $\mu_{var}(P, x_2) \leq H(\mathcal{U}_{|x_2}) = 8\,\text{bits} \leq \Psi(x_2)$, P complies to information flow policy Ψ.

6.4 Policy-Aware Security and Performance Trade-off Analysis

This section presents the PASAPTO optimization problem. In Section 6.4.1, we first formalize the problem of policy-aware security and performance trade-off (PASAPTO) analysis as an optimization problem and prove the NP-hardness of the corresponding decision problem. Sections 6.4.2 and 6.4.3 then present a greedy heuristic and a genetic algorithm efficiently approximating a solution to the optimization problem.

6.4.1 The PASAPTO Optimization Problem

If a program P violates a quantitative information flow policy Ψ, then there exists some $v \in V$ such that the adversarial information flow exceeds the defined threshold $\Psi(v)$. In order for a program variant P' to be compliant with Ψ, the information flow with regards to v must be decreased. This can be achieved by removing control-flow statements involving data from v. To this end, we introduced the syntax of a control-flow removal algorithm $\mathcal{T}$ (Definition 48) in Section 6.2.3. The purpose of $\mathcal{T}$ is, however, not limited to establishing policy-compliance[1]. Given a control-flow removal algorithm $\mathcal{T}$, we can also explore the trade-off between security and performance (see Definition 51). More precisely, each binary vector $t \in \{0,1\}^{|T|}$ can be interpreted as a specific selection of such a trade-off. The problem of finding those transitions corresponding to a policy-compliant program with Pareto-optimal security and performance can then be expressed as an optimization problem with objective function

$$f(t) := \begin{pmatrix} \mu_s(\mathcal{T}(P,t)) \\ \mu_p(\mathcal{T}(P,t)) \end{pmatrix} .$$

In more detail, we are interested in finding those binary vectors $t \in \{0,1\}^{|T|}$ for which the cost function $f(t)$ is minimal and the program $\mathcal{T}(P,t)$ complies with Ψ. Formally, this can be expressed using the argument minimum function:

$$\begin{aligned} \arg\min_{t} \quad & f(t) \\ \text{subject to} \quad & \mathcal{T}(P,t) \text{ is compliant with } \Psi, \\ & t \in \{0,1\}^{|T|} \end{aligned} \tag{6.3}$$

Since f has multiple objectives, solving this optimization problem will in general not yield a single optimal function argument, but a set of Pareto-optimal solutions.

Complexity Analysis To analyze the complexity of the problem, we define its corresponding decision problem and a class of security and performance functions which allow to construct programs with certain behavior.

[1]Note that policy-compliance can always be established since $\mathcal{T}$ can be used to remove all control-flow transitions of a program.

Definition 55 (Polynomial-Time Linear Expandability). *Let μ_{var} be a variable-specific information flow measure and μ_p be a program performance measure. We call (μ_{var}, μ_p)* polynomial-time linear expandable *iff:*

1. *$\forall P \in \mathcal{P}\ \forall n \in \mathbb{N}_0\ \forall v \in V$: we can modify P (in polynomial time) to P' by adding a control-flow decision τ_i such that for $t_i = 1 : \mu_{var}(P', v) = \mu_{var}(P, v) + n$ and $\forall v' \in V, v' \neq v : \mu_{var}(P', v') = \mu_{var}(P, v')$.*

2. *$\forall P \in \mathcal{P}\ \forall n \in \mathbb{N}_0\ \forall \tau_i \in T$: we can modify (in polynomial time) P to P' such that for $t_i = 1 : \mu_p(P') = \mu_p(P) + c$ and for $t_i = 0 : \mu_p(P') = \mu_p(P) + 2c$.*

3. *For the empty program P_0 it applies that $\mu_p(P_0) = 0$ and $\forall v \in V : \mu_{var}(P_0, v) = 0$.*

Definition 56 (PASAPTO Decision Problem). *Given $P, \Psi, \mu_{var}, \mu_s, \mu_p$ and k_s, $k_p \in \mathbb{N}_0$, the* PASAPTO decision problem *is to decide whether*

$$\exists t \in \{0,1\}^{|T|}$$

such that for the resulting program variant

$$P' = \mathcal{T}(P, t)$$

it holds that

$$P' \textit{ is compliant to } \Psi \wedge \mu_s(P') \leq k_s \wedge \mu_p(P') \leq k_p.$$

Theorem 8 (NP-Hardness of PASAPTO Decision Problem). *If (μ_{var}, μ_p) are polynomial-time linear expandable, then the PASAPTO decision problem is NP-hard.*

Proof Sketch. We show a reduction from the 0-1 integer linear programming (0-1-ILP) decision problem, which is known to be NP-complete. Thereby, every 0-1-ILP variable represents a control-flow decision and every 0-1-ILP condition represents a variable in the program. If a control-flow decision is revealed, the leakage of each variable shall be increased by the corresponding coefficient in the 0-1-ILP conditions. Thus, the policy represents the conditions of the 0-1-ILP and the performance represents the objective function.

Proof. Given $A = (a_{i,j}) \in \mathbb{Z}^{n \times m}$, $b \in \mathbb{Z}^n$, $c \in \mathbb{Z}^m$ and $k \in \mathbb{Z}$, the *0-1 integer linear programming (0-1-ILP) decision problem* is to decide whether there exists an $x \in \{0,1\}^m$ such that $A \cdot x \leq b$ and $c^\top x \geq k$. This problem is known to be NP-complete. We perform a many-one reduction of the PASAPTO decision problem to the 0-1-ILP decision problem by transforming an 0-1-ILP instance I into a PASAPTO instance we denote by $\Gamma(I)$. Starting with the empty program P_0, we modify it as follows:

- For each column j of A we add a variable z_j and two control-flow transitions t_j and $\tilde{t}_j$, which each leak 1 of z_j independently.

- For each row i of A we create a variable y_i. For all $a_{i,j} \in A$, if $a_{i,j} < 0$ then $\tilde{t_j}$ shall leak $-a_{i,j}$ of variable y_i, else t_j shall leak $a_{i,j}$ of variable y_i.
- Let $c_{sum} := \sum_{k=1}^{m} |c_k|$. For all c_j in c, if $c_j < 0$ then each branch of t_j needs $c_{sum} - c_j + 1$ time and each branch of $\tilde{t_j}$ needs $c_{sum} + 1$ time, else each branch of t_j needs $c_{sum} + 1$ and each branch of $\tilde{t_j}$ needs $c_{sum} + c_j + 1$ time.
- Let $\delta : \mathbb{Z} \to \mathbb{N}$ with
$$\delta(z) := \begin{cases} 0, & \text{for } z \geq 0 \\ -z, & \text{for } z < 0 \end{cases}$$
For each row i of A we define the policy $\Psi(y_i) = b_i + \sum_{k=1}^{m} \delta(a_{i,k})$. For each column j of A define the policy $\Psi(z_j) = 1$.

Let $k_p = 3m(c_{sum} + 1) + 2c_{sum} + \min(k, c_{sum})$ and $k_s = \infty$. For $x \in \{0, 1\}^m$ we denote $\tilde{x}$ as vector with the same size as x and $\forall x_j : \tilde{x}_j := 1 - x_j$. We show that the output $\Gamma(I)$ equals the output of I.

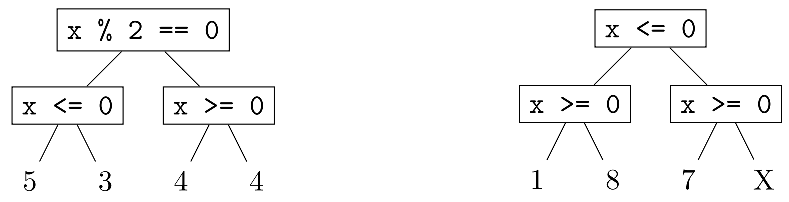
Lemma 1. *For a 0-1-ILP I and its transformation $\Gamma(I) = (P, \Psi)$ it applies that:*

1. $\{x \mid \ A \cdot x \leq b\} = \left\{x \mid \mathcal{T}\left(P, \begin{pmatrix} x \\ \tilde{x} \end{pmatrix}\right) \Psi\text{-compliant}\right\}$
We denote this set as S.
2. $\forall x \in S :$
$$c^\top \cdot x = \mu_p\left(\mathcal{T}\left(P, \begin{pmatrix} x \\ \tilde{x} \end{pmatrix}\right)\right) - 3m(c_{sum} + 1) - 2c_{sum}$$
3. *Let*
$$F := \{(y, z)^\top \mid \mathcal{T}\left(P, (y, z)^\top\right) \Psi\text{-compliant} \wedge y \notin S\}$$
then $\forall x \in S \ \forall f \in F :$
$$\mu_p\left(\mathcal{T}\left(P, \begin{pmatrix} x \\ \tilde{x} \end{pmatrix}\right)\right) < \mu_p\left(\mathcal{T}\left(P, f\right)\right)$$
and
$$\mu_p\left(\mathcal{T}\left(P, f\right)\right) > 3m(c_{sum} + 1) + 3c_{sum}$$

Proof. By construction. □

Lemma 1 shows that the PASAPTO decision problem is many-one reducible to the 0-1-ILP decision problem. This shows that the PASAPTO decision problem is NP-hard. □

Corollary 8.1. *If μ_{var}, μ_s and μ_p can be computed in polynomial time (e.g., constant time), then the PASAPTO decision problem is NP-complete.*



(a) Control flow of original program P (b) Control flow of program variant P'

Figure 6.2: Control-flow graphs for proof of Theorem 9

Heuristic Approaches To overcome the time complexity implications of Theorem 8, we can employ heuristic approaches to efficiently find satisfactory solutions. A heuristic may not necessarily find optimal solutions, but we require any returned solution to satisfy the constraints, i.e., to be policy compliant.

A naïve heuristic could assume that removing a control-flow transition always decreases the adversarial information flow. Formally, this assumption can be described as follows: Consider a partial order

$$\leq_p \subseteq \{0,1\}^{|T|} \times \{0,1\}^{|T|}$$

defined by

$$t \leq_p t' :\Longleftrightarrow \forall i \in [1, |T|] : t_i \leq t'_i \ .$$

Given a program P and a security measure μ_s, the stated assumption then is equivalent to the question whether the function

$$g(t) := \mu_s(\mathcal{T}(P, t))$$

with input $t \in \{0,1\}^{|T|}$ is monotonically increasing. We can show that this assumption does not hold in the general case.

Theorem 9 (Non-Monotonicity of Control Flow Leakage)**.** *Let $\leq_p$ and $g(t)$ be as defined before. Then, $g(t)$ is not monotonic increasing in the variable $t \in \{0,1\}^{|T|}$ for partial order $\leq_p$. Formally, $t \leq_p t' \not\Rightarrow g(t) \leq g(t')$.*

Proof by Contradiction. Consider the following code of program P with input $x \in D_4$ distributed uniformly and the maximum information flow measure μ_{max}.

```
int example3(int x)
    if (x % 2 == 0)
        if (x <= 0)
            x += 1
    else
        if (x >= 0)
            x -= 2
    return x
```

The control flow of program P is illustrated in Figure 6.2a and includes the size of each indistinguishable equivalence class in the leaves of the program path. We obtain a worst-case information flow of

$$\mu_{max}(P) = 4 - \min\{\log 5, \log 3, \log 4\} = 2.42\,\text{bits}.$$

Now consider the program variant

$$P' := \mathcal{T}(P, t') \text{ with } t' := (0,1,1)^T,$$

that is, we apply $\mathcal{T}$ to P in order to hide the control flow of the first transition (`x % 2 == 0`). To this end, P' executes the then-branch as well as the else-branch of the hidden control-flow transition. In P' the adversary can thus observe the control flow of both branches for every input. The control flow of P' is illustrated in Figure 6.2b, where 'X' denotes an infeasible path. The combination of branches leads to an increased information flow of

$$\mu_{max}(P') = 4 - \min\{\log 1, \log 8, \log 7\} = 4 \text{ bits}.$$

Defining $t := (1,1,1)^T$, we can conclude that

$$\begin{aligned} t' &\leq_p t \\ \Rightarrow \mu_{max}(\mathcal{T}(P,t')) &\geq \mu_{max}(\mathcal{T}(P,t)). \end{aligned}$$

and as thus there is no monotonic increasing structure in general. □

Even if there is no general structure, one could find some special cases where the monotony assumption holds. For example, a program without any nested conditions, as well as the restriction of removing only inner conditions of a program with nested ones fulfill this assumption. Our heuristics presented in the remainder of this section implicitly take this knowledge into account.

6.4.2 GreedyPASAPTO: A greedy heuristic

Our first algorithm GreedyPASAPTO is a greedy heuristic providing fast convergence. Starting point of this algorithm is a transformation of P not containing any control-flow transitions. We call this program *the all-hidden program.* We know that this program is compliant with Ψ, because it does not entail any adversarial information flow at all. Based on the all-hidden program, we iteratively reveal control-flow transitions until revealing any other control-flow transition would lead to a non-compliant program or a program dominated with respect to the cost function. By incrementally revealing control-flow transitions, we expect to gradually obtain policy-compliant programs with better performance.

Structure GreedyPASAPTO is structured as follows. It takes as input a program P and a quantitative information flow policy Ψ, and outputs a solution set $\mathbb{P}$ containing non-dominated program variants of P. In each iteration step, we consider a base program, starting with the all-hidden program in the first step, and a bit vector set B corresponding to programs with one additional control-flow transition revealed. We filter any policy-compliant and non-dominated program and add its corresponding bit vector to the current bit vector set B. The algorithm terminates if every program is non-compliant or dominated by a program of the solution set or if there are no more transitions to reveal. At the end of each iteration, all program variants corresponding to a vector in B are added to the solution set. One of these programs is randomly chosen as a base for the next iteration step. Filtering non-dominated programs is achieved by the subroutine *filterNonDominated*$(\cdot)$ that on input a set of programs outputs the maximum subset of non-dominated programs. The details of GreedyPASAPTO are presented in Algorithm 1.

Algorithm 1 GreedyPASAPTO: Greedy Heuristic

```
Require: $P$ – Program under inspection
    $\Psi$ – Quantitative information flow policy
Ensure: $\mathbb{P}$ – Non-dominated set of program variants of $P$
procedure GreedyPASAPTO($P, \Psi$)
    $T := \mathit{filterControlFlow}(P)$
    $V := \mathit{computeVariableSpace}(P)$
    $B := \{0_{|T|,1}\}$
    $\mathbb{P} := \emptyset$
    while $B \neq \emptyset$ do
        $t \leftarrow_{\$} B$
        $B := \emptyset$
        for all $i \in \{n \in \mathbb{N} \mid 1 \leq n \leq |T|, t_n = 0\}$ do
            $t' := t + e_i$
            if $\forall v \in V : \mu_{var}(\mathcal{T}(P, t'), v) \leq \Psi(v)$ then
                $B := B \cup \{t'\}$
            end if
        end for
        $\mathbb{P} := \mathit{filterNonDominated}(\mathbb{P} \cup \{\mathcal{T}(P, t) \mid t \in B\})$
        for all $t' \in \{t \in B \mid \mathcal{T}(P, t) \notin \mathbb{P}\}$ do
            $B := B \setminus \{t'\}$
        end for
    end while
    return $\mathbb{P}$
end procedure
```

Alternative approach We prefer the approach of starting with the all-hidden program and revealing control-flow transitions over the alternative of starting with the all-revealed (i.e., the original) program and removing control-flow transitions for two reasons. First, the all-hidden program is guaranteed to be policy-compliant. The alternative approach on the other hand has to somehow establish policy-compliance by investigating other program variants. In doing so, a large number of non-compliant programs may have to be investigated. Second, one could think that in the alternative approach removing additional control flow from a policy-compliant program would always lead to another policy-compliant program and hence policy-compliance does not have to be checked again. However, Theorem 9 shows that this is not true in the general case and hence policy-compliance has to be checked for each candidate program.

6.4.3 GeneticPASAPTO: A genetic algorithm

GeneticPASAPTO is a heuristic approach for solving optimization problem (6.3) based on a genetic meta-heuristic. Genetic algorithms do not require any a priori knowledge about the structure of the search space, thus they fit our problem very well. In contrast to our greedy heuristic, a whole set of not necessarily policy-compliant solutions, the so-called population, is considered and used to generate new solutions. GeneticPASAPTO selects the fittest individuals, i.e. binary vectors of size $|T|$, from the population according to some fitness function. Based on the selected individuals, by using so-called crossing and mutation, new individuals are generated which replace the least fittest individuals in the population. This procedure is repeated until a sufficiently large amount of non-dominated solutions have been found or a running time bound has been reached.

Our genetic algorithm uses a population size of N determined by the developer. The algorithm outputs for a program P and a quantitative information flow policy Ψ a non-dominated solution set of policy-compliant program variants of size at most N. Since genetic algorithms may converge to one solution, to obtain a wide selection of solutions for the developer, GeneticPASAPTO uses niching methods [Mah95]. The details of GeneticPASAPTO are presented in Algorithm 2.

Testing the dominance is implemented by the subroutine $\textit{dominates}(t_1, t_2)$ which on input two binary vectors outputs $\top$ if the program $\mathcal{T}(P, t_1)$ dominates the program $\mathcal{T}(P, t_2)$ and $\bot$ otherwise. Dominance is defined in terms of the objective function $f(t)$ given by the optimization problem. To simplify the depiction we use the function $\textit{updateFitness}(t_i, t_j)$, which reduces the fitness of individual j by 1 if $\textit{dominates}(t_i, t_j) = \top$. By $J_{m,n}$ we denote the $m \times n$ matrix with all entries equal to one.

Fitness function Our fitness function F is based on a ranking [FF+93] which takes policy-compliance into account. To an individual i we assign $F_i := N - k$ if it is dominated by k individuals in the current population. If a program is not policy-compliant, we assign $F_i := 0$ to penalize such solutions and prefer complying programs.

Crossing and Mutation In the context of genetic algorithms each component of an individual is called a gene. We cross two individuals by switching the first half of the genes of the parents. For those individuals, mutation is applied with a probability of $\frac{1}{|T|}$ for each gene, i.e. the gene is inverted.

Niching We use Sharing [GR+87, pp. 41–49][Hol+92] as our niching method, because it is recommended for multi-objective optimization [Mah95, p. 84]. If two individuals of a population are in the same niche, i.e., their distance is below a certain threshold σ called the sharing parameter, their fitness is shared. The sharing parameter σ should be chosen carefully. An estimation of a good selection of σ based on the bounds of the search space has been made by Fonseca and

Fleming [FF+93]. Applied to our problem, we obtain the unambiguous solution

$$\sigma = \frac{M_1 + M_2 - (m_1 + m_2)}{N - 1}.$$

We approximate the bounds of the search space using the properties of two well-known programs. We expect the all-revealed program to have high leakage but good performance. On the other hand, we expect the all-hidden program to have no leakage but bad performance.

By determining the distance of two points, we do not want to weight the influence of one dimension over another, because they may have different size scales. To this end, we standardize both dimensions with respect to the maximum values of the programs above. This leads to the following choice of parameters:

$$M_1 := \frac{\mu_s(P)}{\mu_s(P)} = 1 \qquad m_1 := \frac{\mu_s(\hat{P})}{\mu_s(P)} = 0$$

$$M_2 := \frac{\mu_p(\hat{P})}{\mu_p(\hat{P})} = 1 \qquad m_2 := \frac{\mu_p(P)}{\mu_p(\hat{P})},$$

where $\hat{P} := \mathcal{T}(P, (0, \dots, 0)^T)$ represents the all-hidden program. In the following, we denote the scale factor by

$$S := \left(\frac{1}{\mu_s(P)}, \frac{1}{\mu_p(\hat{P})} \right)^T.$$

The fitness is now shared with other individuals in the same niche. We define a sharing function [GR+87, p. 45],

$$sh : [0, \infty) \to [0, 1]$$

with

$$sh(d) := \begin{cases} 1 - \frac{d}{\sigma}, & \text{if } d < \sigma \\ 0, & \text{otherwise} \end{cases}$$

where d describes a metric of two points in the search space. We use the Euclidean metric in the following.

Based on the simple fitness function F we can now define a shared fitness function F'. F' takes as arguments an individual i and a matrix M which contains the individuals of a population as columns.

$$F'(i, M) := \frac{F_i}{\sum_{j=1}^{N} sh(d(f(M_{:,i}) \circ S, f(M_{:,j}) \circ S))}$$

Convergence Based on the shared fitness function the evolution process is repeated until the maximum number of iterations (user-defined parameter X) is reached or the convergence criterion is fulfilled. We use *maximum allowable pareto percentage* as our convergence criterion, i.e., the algorithm terminates if the percentage of non-dominated individuals in the current population exceeds the user-defined threshold α. By default, we use $\alpha = 0.7$. The quality of the solution set depends on carefully chosen parameters α and X.

Algorithm 2 GeneticPASAPTO: Genetic Algorithm

```
Require: $P$ – Program under inspection
    $\Psi$ – Quantitative information flow policy
    $N$ – Population size
    $\alpha$ – Maximum allowable Pareto percentage
    $X$ – Maximum number of iterations
Ensure: $\mathbb{P}$ – Non-dominated set of compliant programs
 1: procedure GeneticPASAPTO($P, \Psi, N, \alpha, X$)
 2:     $T := \mathit{filterControlFlow}(P)$
 3:     $V := \mathit{computeVariableSpace}(P)$
 4:     $R := 0_{|T|,N}$
 5:     /* Compute share parameter */
 6:     $\hat{P} := \mathcal{T}(P, 0_{|T|,1})$
 7:     $\sigma := \frac{2\mu_p(\hat{P}) - \mu_p(P)}{\mu_p(\hat{P})(N-1)}$
 8:     /* random start selection */
 9:     for $i := 1$ to $N$ do
10:         $R_{i,:} \leftarrow_{\$} \{0,1\}^{|T|}$
11:     end for
12:     /* initial fitness */
13:     $F := J_{N,1}$
14:     $F' := J_{N,1}$
15:     for $i := 1$ to $N$ do
16:         $F_i := N$
17:         if $\exists v \in V : \mu_{var}(\mathcal{T}(P, R_{:,i}), v) > \Psi(v)$ then
18:             $F_i := 0$
19:         else
20:             for $j := 1$ to $N$ do
21:                 $\mathit{updateFitness}(R_{:,j}, R_{:,i})$
22:             end for
23:             $F'_i := F'(i, R)$
24:         end if
25:     end for
26:     $\mathit{paretoPercentage} := 0$
27:     $\mathit{counter} := 1$
28:     while $\mathit{paretoPercentage} \leq \alpha \wedge \mathit{counter} \leq X$ do
29:         /* determine fittest and unfittest */
30:         $\mathit{fittestInd} := \arg\max_{i \in N} F'_i$
31:         $\mathit{fittest} := R_{:,\mathit{fittestInd}}$
32:         $\mathit{secondFittestInd} := \max_{i \in N \setminus \{\mathit{fittestInd}\}} F'_i$
33:         $\mathit{secondFittest} := R_{:,\mathit{secondFittestInd}}$
34:         $a := \arg\min_{i \in N} F'_i$
35:         $b := \arg\min_{i \in N \setminus \{a\}} F'_i$
```

```
/* crossing */
$\mathit{firstChild} := \mathit{fittest}$
$\mathit{secondChild} := \mathit{secondFittest}$
**for** $i := 1$ **to** $\lfloor \frac{|T|}{2} \rfloor$ **do**
    $\mathit{firstChild}_i := \mathit{secondFittest}_i$
    $\mathit{secondChild}_i := \mathit{fittest}_i$
**end for**
/* mutation */
$\mathit{randomVector} \leftarrow_{\$} \{n \in \mathbb{N} \mid 1 \leq n \leq N\}^{|T|}$
**for** $i := 1$ **to** $|T|$ **do**
    **if** $\mathit{randomVector}_i = 1$ **then**
        $\mathit{firstChild}_i := \mathit{firstChild}_i \oplus 1$
    **end if**
    **if** $\mathit{randomVector}_i = N$ **then**
        $\mathit{secondChild}_i := \mathit{secondChild}_i \oplus 1$
    **end if**
**end for**
**for** $i := 1$ **to** $N$ **do**
    /* update after removing old */
    **if** $\mathit{dominates}(R_{:,a}, R_{:,i}) = \top$ **then**
        $F_i := F_i + 1$
    **end if**
    **if** $\mathit{dominates}(R_{:,b}, R_{:,i}) = \top$ **then**
        $F_i := F_i + 1$
    **end if**
    /* update after inserting new */
    $\mathit{updateFitness}(\mathit{firstChild}, R_{:,i})$
    $\mathit{updateFitness}(\mathit{secondChild}, R_{:,i})$
    $F'_i := F'(i, R_{:,\cdot})$
**end for**
/* Reinitialize */
$R_{:,a} := \mathit{firstChild}$
$R_{:,b} := \mathit{secondChild}$
$F_a := N$
$F_b := N$
```

```
        /* fitness of new */
        for $i := 1$ to $N$ do
            $updateFitness(R_{:,i}, R_{:,a})$
            $updateFitness(R_{:,i}, R_{:,b})$
        end for
        if $\exists v \in V : \mu_{var}(\mathcal{T}(P, R_{:,a}), v) > \Psi(v)$ then
            $F_a := 0$
        end if
        if $\exists v \in V : \mu_{var}(\mathcal{T}(P, R_{:,b}), v) > \Psi(v)$ then
            $F_b := 0$
        end if
        $F'_a := F'(a, R)$
        $F'_b := F'(b, R)$
        /* update loop */
        $n := 0$
        for $i := 1$ to $N$ do
            if $F_i = N$ then
                $n := n + 1$
            end if
        end for
        $paretoPercentage := \frac{n}{N}$
        $counter := counter + 1$
    end while
    /* Prepare Return */
    $\mathbb{P} := \emptyset$
    for $i := 1$ to $N$ do
        if $F_i = \max_j F_j$ then
            $\mathbb{P} := \mathbb{P} \cup \{\mathcal{T}(P, R_{:,i})\}$
        end if
    end for
    return $\mathbb{P}$
end procedure
```

6.5 A PASAPTO Analysis for Dataflow Authentication

In this section, we describe a PASAPTO analysis for dataflow authentication (cf. Chapter 5). This combination of DFAuth and PASAPTO forms the basis of our dataflow authentication trade-off analyzer (DFATA) described in Section 6.6 and evaluated in Section 6.7.

In the remainder of this section, we first explain why PASAPTO can be applied to DFAuth. Then, we present potential instantiations of security measures performance measures and control-flow removal algorithms.

Compatibility Recall from Section 5.2.1 that DFAuth considers an active adversary controlling the cloud server, who can (i) read and modify the contents of all variables and the program text (except in the trusted module) (ii) observe and modify the control flow (except in the trusted module) (iii) do all of this arbitrarily interleaved. The security guarantee of DFAuth is to reveal only the information about the inputs to the untrusted cloud server that can be inferred from the program's executed control flow.

Inherently, DFAuth considers an active adversary, but due to its security guarantees any adversary is limited to passively observing the control flow of the program executed on the cloud server. Now recall from Section 6.2.2 that PASAPTO considers a passive adversary capable of continuously observing the control flow of an execution of a program. As such, we can extend DFAuth with a PASAPTO analysis to further reduce the adversarial information flow.

Security and Performance Measures In Section 6.3.1 we presented two program security measures capturing the leakage resulting from control-flow observations. Both, average information flow ($\bar{\mu}$) and maximum information flow (μ_{max}), can be used to instantiate the program security measure μ_s in our DFATA analysis, depending on the information to be captured. To determine compliance of a program with a quantitative information flow policy, we use the variable-specific information flow measure (μ_{var}) introduced in Section 6.3.2.

We consider two instantiations of the program performance measure μ_p. The running time performance measure $\hat{\mu}$ captures the elapsed wall-clock time of the execution of a DFAuth-transformed program. An alternative program performance measure can be defined by applying program analysis techniques to count the instructions used by the program and assign each DFAuth operation a cost value. However, this measure is outside of the scope of this work.

Control Flow Removal Algorithm We consider two instantiations of $\mathcal{T}$ which we refer to as straightlining ($\mathcal{T}_s$) and oblivious state update ($\mathcal{T}_o$). $\mathcal{T}_s$ rewrites code fragments containing conditional instructions into semantically equivalent code not containing any control-flow transitions. For example, bitwise operations can be combined with a constant-time conditional assignment operation to avoid conditional instructions [Mol+05]. $\mathcal{T}_o$ executes both the *then* and the *else* branch of a conditional [Aga00; RLT15]. $\mathcal{T}_o$ forks the state of the program before branching and obliviously updates the program state with the correct one. To this end, we extend DFAuth with an additional trusted module invocation. Provided with the two state variants and the conditional, the trusted module determines the correct variant and returns it to the cloud server. Before returning, the correct variant is re-randomized such that the untrusted part of the server remains oblivious as to which of the two states was returned. Note that this extension is in line with DFAuth's goal of a program-independent trusted code base.

6.6 Implementation

In this section, we present details about DFATA, our Java-based dataflow authentication trade-off analyzer. At the core of our implementation is Soot, a framework for analyzing and manipulating Java programs [Lam+11]. Soot is used to implement our own DFAuth compiler, detection of control-flow leaks and control-flow removal. We implement the DFAuth trusted module in an SGX enclave.

We compute the indistinguishability partition required to quantify control-flow leakage based on JBSE (Java Bytecode Symbolic Executor) [BDP13; BDP15]. Using symbolic execution we obtain the set of symbolic paths of a program and the corresponding path conditions. By construction, the path conditions are equal to the constraints of each equivalence class. We assume the random variable $\mathcal{U}$ to be distributed uniformly and transform the constraints into a system of linear inequalities for each equivalence class. The numeric evaluation of each equivalence class is determined using LattE [DL+04]. LattE implements Barvinok's algorithm [Bar94] to compute the size of the equivalence classes exactly and in polynomial time.

DFATA supports our two heuristics as well as an exhaustive search trade-off analysis algorithm. DFATA takes Java Bytecode operating on plaintext data and an associated information flow policy as input. It outputs the result of the trade-off analysis as a set of fully executable programs operating on encrypted data using DFAuth.

DFATA currently operates only on a subset of Java. It performs intra-procedural analysis and does not handle exceptions, because it is limited by its tools – DFAuth and the QIF analysis. Extensions are possible, but orthogonal to our work on heuristics. For an overview on techniques, tools and trade-offs for model counting involving non-linear numeric constraints, we refer to a survey by Borges et al. [Bor+17]. We refer to Aydin et al. [Ayd+18] for how to perform model counting on string (sequences of characters) constraints and mixed string and integer constraints. For multi-instantiation of a class in a sensitive and an insensitive context, we refer to Dong et al. [DMD16]. Also, the current implementation is not optimized for performance, but for the evaluation of the quality of our heuristics.

6.7 Evaluation

In this section, we present the results collected in two experiments in which we applied DFATA. In the first experiment (presented in Section 6.7.1), we consider an implementation of an electronic auction with sealed bids. In the second experiment (presented in Section 6.7.2), we inspect a program performing decision tree evaluation on medical data. We chose these programs such that we are able to explore all program variants in order to evaluate the effectiveness of our heuristics, but complex enough to be non-trivial in the context of computation on encrypted data (e.g., FHE or MPC).

In each experiment, we first use DFATA in exhaustive mode to obtain all program variants as well as their average information flow ($\bar{\mu}$) and running time performance ($\hat{\mu}$). We determine $\hat{\mu}$ by executing each variant multiple times on random inputs, then computing the mean over all measurements. From these results, we determine the Pareto front of the PASAPTO optimization problem (modulo policy-compliance). We then execute each of our two probabilistic heuristics multiple times.

In each run, we compute the hypervolume indicator (HVI) (cf. Section 2.4.2) of the heuristic solution set with respect to the dimensions $\bar{\mu}$ and $\hat{\mu}$. We evaluate the quality of our algorithms using the relative difference between the HVI of a heuristic solution set and the HVI of the Pareto front:

$$\mathrm{HVI}_{\mathrm{rel}} := \frac{\mathrm{HVI}_{\mathrm{heuristic}}}{\mathrm{HVI}_{\mathrm{Pareto\ front}}} - 1.$$

Elements of high quality solution sets according to $\mathrm{HVI}_{\mathrm{rel}}$ are close to the Pareto front and cover a wide range of trade-offs.

For each experiment and heuristic, we perform 101 runs. We perform an odd number of runs such that we can unambiguously identify a single run as the *median run* according to $\mathrm{HVI}_{\mathrm{rel}}$. Scatter plots presented in the following show data points for this particular run. In addition to the median $\mathrm{HVI}_{\mathrm{rel}}$, we present the 95th percentile ($Q_{0.95}$) of $\mathrm{HVI}_{\mathrm{rel}}$ over all runs.

Because our implementation of DFATA is not optimized for performance, but for the evaluation of the quality of the heuristics, we cannot provide wall-clock running times of our heuristics. However, we can estimate the total running time t of a heuristic by $t = v * (s + p) + h$ where v refers to the number of program variants visited, s and p denote the time to compute $\bar{\mu}$ respectively $\hat{\mu}$ of a program variant, and h refers to the running time of the heuristic itself.

All experiments were conducted in the Microsoft Azure Cloud using Azure Confidential Computing VM instances of type `Standard_DC4s`. Each instance runs Ubuntu Linux 18.04 and has access to 4 cores of an SGX-capable Intel Xeon E-2176G CPU and 16 GB RAM.

6.7.1 Electronic Sealed-Bid Auction

In a sealed-bid auction, individual bids $\{h_0, \ldots, h_{n-1}\}$ must be kept confidential. The winner of the auction is identified by the highest bid and is announced publicly.

Experimental Setup Consider the following pseudocode implementing such an auction.

```
int auction(int[] h)
    int l = 0;
    for (int i = 0; i < h.length; i++)
        if (h[i] > h[l])
            l = i;
    return l;
```

On input an array h containing n bids, the program determines the identity of the winner l as the array index of the winning bid.

A similar piece of code was previously investigated by Backes et al. [BKR09], although in a different context. They considered a secret (high-sensitive) input array h and allowed the adversary to only observe the public (low-sensitive) output l. In our case, the adversary is allowed to observe the control flow of the program during execution.

The input array h is ordered randomly to protect the identities of the bidders. In doing so, we also ensure a fair determination of the winner in case the highest bid does not unambiguously identify a winner.

In this experiment, we assume $n = 10$. After loop unrolling, the resulting program performs $n-1$ comparisons, respectively control-flow transitions. Since one can decide whether to *reveal* or *hide* for each transition, our algorithms operate on a search space containing $2^{n-1} = 512$ program variants. For GeneticPASAPTO we use a population size of $N := 2 \cdot |T| = 19$ and a bound of $X := 10 \cdot |T| = 90$.

Evaluation Results Figure 6.3 shows the median leakage and running time of all possible program variants grouped by their number of hidden control-flow transitions. The number of variants per aggregation group is $\binom{n-1}{k}$ where k is the number of hidden control-flow transitions. For example, the search space contains $\binom{9}{2} = 36$ variants with 2 hidden transitions. The chart experimentally confirms the negative correlation between leakage and running time we expect.

Figure 6.4 relates the behavior of the median run of GreedyPASAPTO to the entire search space. Figure 6.5 does the same for GeneticPASAPTO. Each plot contains the entire search space of 512 program variants. The distinguished all-hidden and all-revealed programs are highlighted. The set of Pareto-optimal programs is a subset of the search space and contains elements of the Pareto front of the PASAPTO optimization problem. The set of *visited* programs is a subset of the search space and contains programs which have been investigated by the algorithm. Points marked as *solution* are part of the solution set output by the algorithm.

As is expected, no program with worse performance than the all-hidden program exists. One would expect the all-revealed program to provide the best performance, but DFATA found another one with better performance. However, we believe this to be only due to the inaccurate nature of the running time measurement. The performance range over all program variants reaches from 7.84 ms for the best performing program to 15.63 ms for the all-hidden program.

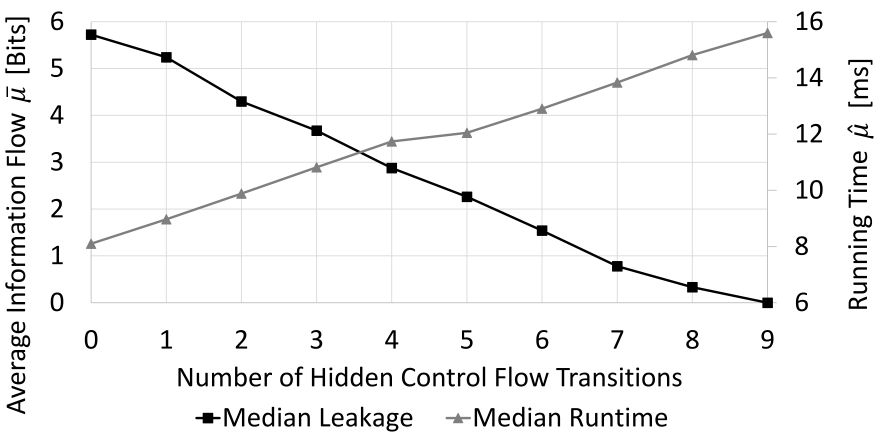


Figure 6.3: Median leakage (left y-axis) and median running time (right y-axis) of the set of all Auction program variants grouped by number of hidden control-flow transitions.

The security range reaches from a leakage of 0 bits for the all-hidden program to 5.723 bits for the all-revealed program.

In Figure 6.4 we can see that GreedyPASAPTO visited only 46 points and output 18 points in the solution set. The algorithm still found a solution close to the Pareto front. In the median, the heuristic solution is $\mathrm{HVI}_{\mathrm{rel}} = 5.35\%$ ($Q_{0.95} = 19.18\%$) worse than the Pareto front.

In Figure 6.5 we can see that GeneticPASAPTO also produced a solution close to the Pareto front, but visited 38 programs and output 13 in the solution set. For the genetic algorithm, we obtain $\mathrm{HVI}_{\mathrm{rel}} = 19.47\%$ with $Q_{0.95} = 30.36\%$. Even if the quality of the median run is worse than the quality of GreedyPASAPTO, we can repeat the execution of GeneticPASAPTO to obtain a higher quality since the algorithm is probabilistic. Moreover, the advantage of GeneticPASAPTO is that it can potentially find any solution, whereas GreedyPASAPTO could miss some solutions in every run due to its design.

In this experiment, measuring the QIF of a program variant approximately takes time $s = 26\,\mathrm{s}$ and measuring the performance of a program variant approximately takes time $p = 12\,\mathrm{ms}$. In conclusion, the running time would be roughly $t = 1200\,\mathrm{s}$ for GreedyPASAPTO ($h = 0.1\,\mathrm{ms}$), and $t = 1000\,\mathrm{s}$ for GeneticPASAPTO ($h = 6\,\mathrm{ms}$).

6.7.2 Decision Tree Evaluation

In this experiment, we use DFATA on a program performing decision tree evaluation on sensitive medical data. Consider the use case of a research institution providing a decision tree evaluation service to medical institutions such as hospitals. Medical institutions can submit patient's health data and obtain the result of the decision tree classification to aid their diagnosis. Since patient data is highly confidential, the research institution has to compute on encrypted data using DFAuth.

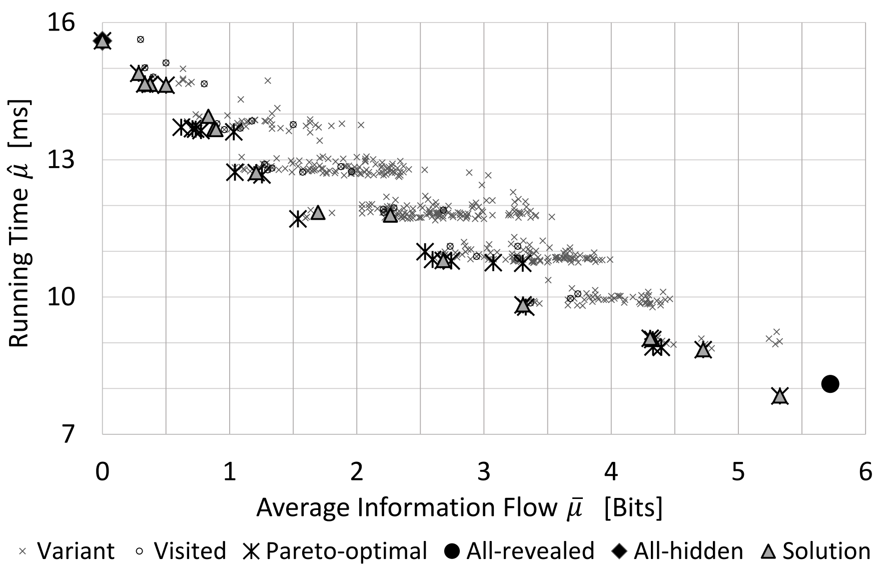


Figure 6.4: Greedy algorithm applied to Auction Program.

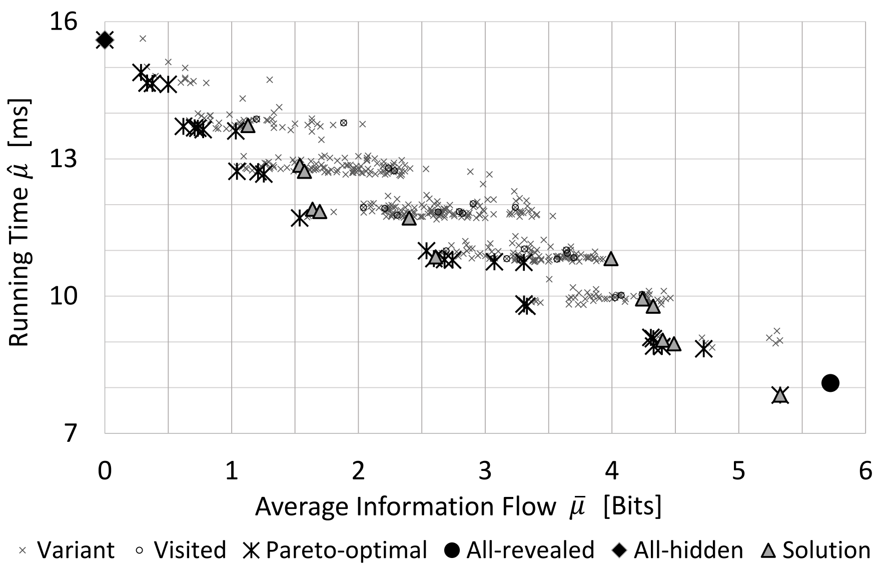


Figure 6.5: Genetic algorithm applied to Auction Program.

The security goal is to protect the sensitive inputs to the decision tree. The sensitivity of each input is provided as a quantitative information flow policy. The protection of the output of the decision tree is not a security goal and may be learned by the decision tree service provider.

Experimental Setup We consider a program making predictions about breast cancer based on the decision tree by Sumbaly et al. [SVJ14]. Provided with medical information, the program outputs a prediction of whether or not the patient has breast cancer. The pruned decision tree takes six input variables and performs 13 control-flow decisions. Thus, the size of the search space is $2^{13} = 8192$. The attribute represented by each variable is described in Table 6.1.

Variable	Attribute	$\Psi(v_i)$
v_1	Clump Thickness	2
v_2	Uniformity Cell Size	∞
v_3	Uniformity Cell Shape	∞
v_4	Marginal Adhesion	1
v_5	Bare Nuclei	∞
v_6	Bland Chromatin	0

Table 6.1: Overview on variables and our QIF policy

Input variables are integers in the interval $[1, 10]$, which our leakage model captures accordingly. The information of each variable is $\log(10) = 3.32$. Our analysis can take into account that not all variables are equally sensitive. Under the assumption that (i) the bland chromatin attribute is critically sensitive and no information about it must leak, (ii) the clump thickness attribute is highly sensitive, (iii) the marginal adhesion attribute is sensitive and (iv) all other variables are not sensitive, a developer may define a quantitative information flow policy as presented in the third column of Table 6.1. We will assume this policy in our experiment. For GeneticPASAPTO we use a population size of $N := 4 \cdot |T| = 52$ and a bound of $X := 20 \cdot |T| = 260$.

Evaluation Results Figures 6.6 and 6.7 present results concerning the search space, i.e., all program variants. Figures 6.8 and 6.9 present the results of our two heuristic algorithms.

Figure 6.6 shows leakage depending on the number of hidden control-flow transitions. For k hidden transitions, $\binom{13}{k}$ programs are aggregated. In contrast to the previous experiment (see Figure 6.3), we can see that the median leakage does not decrease before more than 7 transitions are hidden. The mean leakage even increases in comparison to the all-revealed program. This highlights that as per Theorem 9 leakage can increase when removing control-flow transitions.

Figure 6.7 shows the running time depending on the number of hidden transitions. In particular, the minimum of aggregated variants shows that up to 8 transitions can be hidden without significantly increasing the running time. The analysis shows that the leakage and running time of complex programs is not linear in the number of hidden transitions and it is important to find combinations of hidden transitions that barely increase the running time, but significantly reduce leakage.

Figures 6.8 and 6.9 present the results of the median runs for each of our two algorithms. Note that for this experiment, we do not show the data points of all 8192 program variants, but only those which are policy-compliant or have been

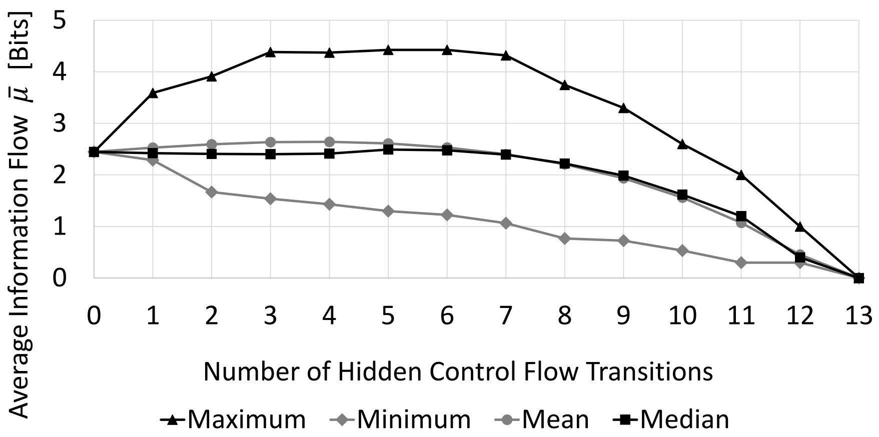


Figure 6.6: Median leakage of the set of all Decision Tree program variants grouped by number of hidden control-flow transitions.

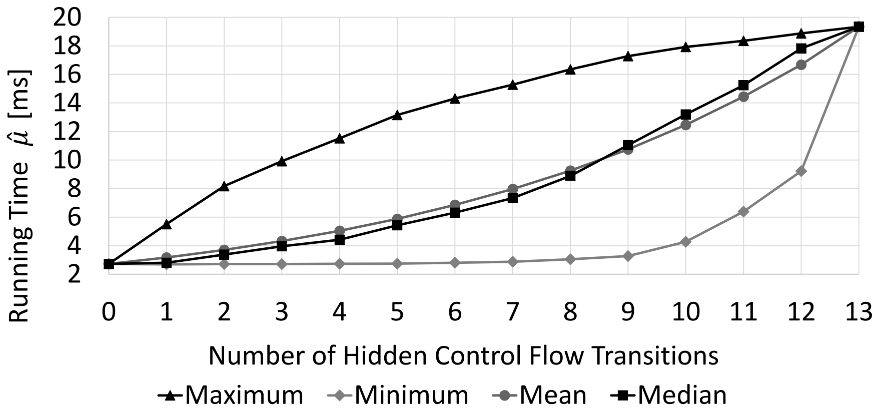


Figure 6.7: Median running time of the set of all Decision Tree program variants grouped by number of hidden control-flow transitions.

visited by our heuristic. The performance range over all policy-compliant program variants reaches from 4.231 ms for the best performing program to 19.341 ms for the all-hidden program. The security range reaches from a leakage of 0 bits for the all-hidden program to a program variant with 3.358 bits leakage. This is even higher than the leakage of the non-compliant all-revealed program with 2.447 bits and additionally has a worse performance of 7.891 ms compared to 2.720 ms. This experimentally confirms Theorem 9, which says that the adversarial information flow might increase when removing control-flow.

As in the auction experiment, GreedyPASAPTO only evaluated a few variants and still produced a solution close to the Pareto front. The algorithm inspected 86 program variants and output 14 programs in the solution set. Visited points that are better than the Pareto front are not compliant with our policy and are not added to the solution set. For this algorithm, we have $\mathrm{HVI}_{\mathrm{rel}} = 1.76\%$ with

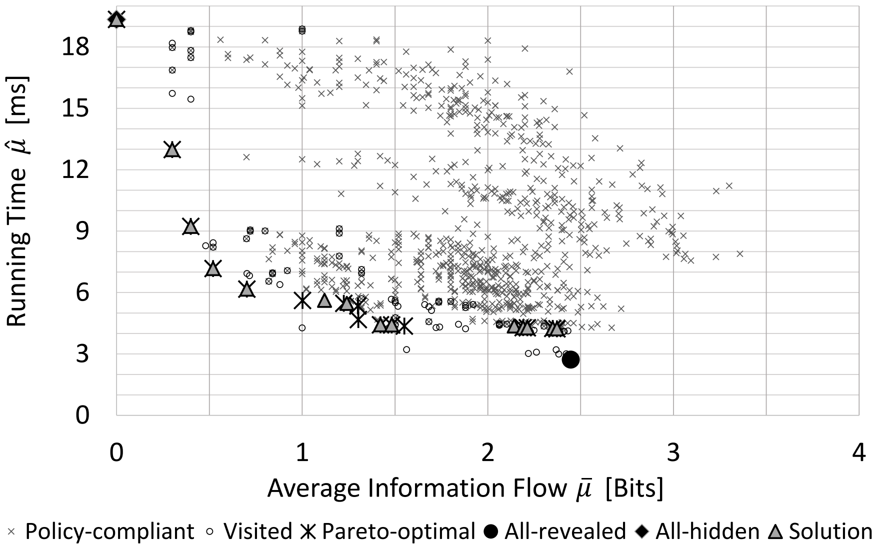


Figure 6.8: Greedy algorithm applied to Decision Tree Program

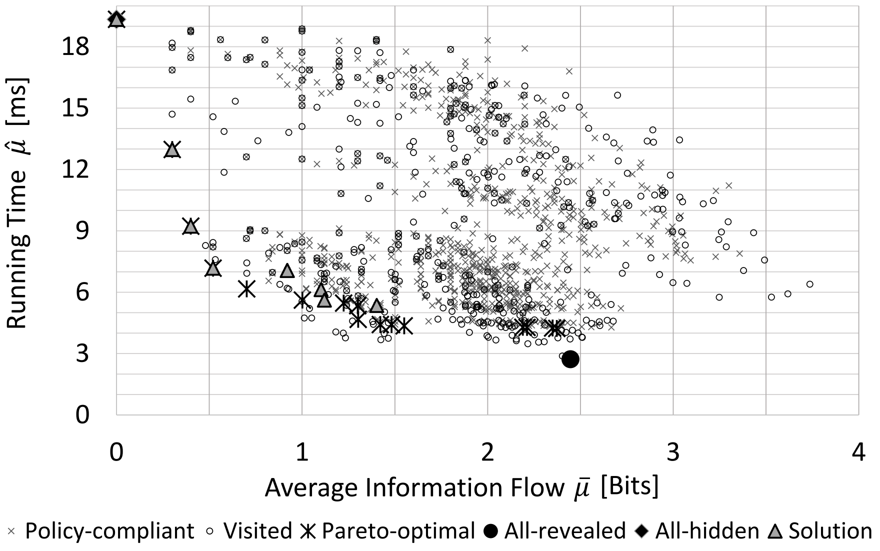


Figure 6.9: Genetic algorithm applied to Decision Tree Program

the 95th percentile $Q_{0.95} = 3.57\%$. The best running time for a policy-compliant program is 4.231 ms, which is also the best running time on the Pareto front.

GeneticPASAPTO visited 415 program variants and output 11 programs in the solution set. Recall that a design goal of this algorithm is the diversity of the elements in the solution set, which is implemented using Niching. For example, the solution set of GreedyPASAPTO contains clusters of non-dominated solutions, e.g., a lot of program variants with running time between 4 and 5 ms with a large difference in leakage but only little difference in performance. In contrast, Niching avoids clustering and outputs a solution set with sufficiently differing values to simplify the choice of the developer. This can lead to a lower quality solution, because it can cause the fitness of two optimal points to be shared resulting in both being possibly removed from the population. For this algorithm, we have $\mathrm{HVI}_{\mathrm{rel}} = 14.08\%$ with the 95th percentile of $Q_{0.95} = 33.80\%$. The best running time found for a policy-compliant program is 5.353 ms which is 26.5% worse than the best running time on the Pareto front. The second experiment provides better solutions than the first one. We believe this is caused by the fact that the second experiment has more structure that can be used by the heuristics.

In this experiment, measuring the QIF of a program variant approximately takes time $s = 15\,\mathrm{s}$ and measuring the performance of a program variant approximately takes time $p = 8\,\mathrm{ms}$. In conclusion, the running time would be roughly $t = 1250\,\mathrm{s}$ for GreedyPASAPTO ($h = 0.5\,\mathrm{ms}$), and $t = 6200\,\mathrm{s}$ for GeneticPASAPTO ($h = 3\,\mathrm{s}$).

Since the running time is dominated by the QIF analysis, we note that a further, possibly very effective optimization is to update the QIF measure, instead of re-computing it from scratch. We do not have an estimate how effective that optimization would be.

6.8 Related Work

The work presented in this chapter is related to the trade-off between security and performance, language-based information flow security and (control flow) side-channels and defenses.

Trade-off between Security and Performance This trade-off has for example been explored in the context of secured networked control systems [ZC11], cyber-physical systems [Zha+16], block ciphers [Wei+13], timing attacks in cryptographic code [DK15] and range queries on encrypted data [LO05]. Wolter et al. [WR10] present a generic security-performance trade-off model based on generalized stochastic Petri nets. However, their model assumes that recovery from an insecure system state is possible. We do not think that recovery from adversarial information flows is possible, hence their model cannot be applied in our case.

Once a specific trade-off has been chosen, formal verification techniques [LHS13; Mol+05; Alm+18] can be used to ensure that compilers do not introduce additional side-channels. However, these techniques are orthogonal to our work since they verify the compliance of an implementation to a given specification (i.e., ideal functionality in MPC) whereas we modify the specification by altering the security-performance trade-off.

Language-Based Information Flow Security For an overview on this subject, we refer to a survey by Sabelfeld et al. [SM03].

The program security measures considered in this work are based on the concept of QIF and inspired by the work of Backes et al. [BKR09]. For foundations of QIF, we refer to Köpf et al. [KB07], Smith [Smi09], Heusser et al. [HM10], Malacaria [Mal11] and Klebanov [Kle14]. Our measures are exact rather than approximate and are based on a two-step QIF analysis consisting of an algebraic interpretation followed by a numeric evaluation. The first step is based on symbolic execution, which has been used previously for information flow analysis [NKP18; OT11; PM14]. Our PASAPTO analysis does not mandate a specific program security measure, but can be instanciated using other measures. For example: Leakwatch [CKN14] estimates leakage of Java programs by repeatedly executing them. Malacaria et al. [Mal+18] consider an approximation of information flow based on noisy side-channels. Kučera et al. [Kuč+17] use the concept of an attacker's belief [CMS05], which orthogonally captures the attacker's accuracy besides uncertainty.

Information flow policies and checking for their compliance has been considered in [Alm+18; HM10; YT11]. Policy establishment has for example been considered in [Alm+18; Kuč+17]. Recently, Kučera et al. [Kuč+17] presented a program synthesis similar to our program transformation. Their approach also takes a program and a policy as inputs and outputs a policy-compliant program. However, their procedure only applies to probabilistic programs and works by adding uncertainty to the program's output while we consider side-channels and preserve the original program's semantics.

(Control-Flow) Side-Channels and Defenses An approach to avoid a broad class of side-channels is to produce constant-time code not making any control-flow decisions on secret data. Molnar et al. [Mol+05] present a source-to-source transformation producing such code by relying on bitwise operations and a constant-time conditional assignment operation. Similarly, Cauligi et al. [Cau+17] define their own domain specific language to produce code adhering to these requirements. Raccoon [RLT15] transactionally executes both branches of a control-flow statement and ensures that the program state is updated obliviously using the correct transaction. GhostRider [Liu+15] defends against side-channels based on memory access pattern by obfuscating programs such that their memory access pattern is independent of control-flow instructions.

6.9 Summary

In this chapter, we showed how to efficiently compute policy-compliant and approximately Pareto-optimal trade-offs between leakage and performance where the decision problem is NP-hard. For a given Java Bytecode program our implementation proposes semantically equivalent, side-channel reduced Java programs for the execution within DFAuth-protected SGX enclaves from which developers can select their desired trade-off. We showed the practical feasibility of this approach for computing on encrypted data in a commercial cloud environment using two example programs. The combined protection by DFAuth (security against active attackers and program-independent enclave code) and PASAPTO (performance-optimized side-channel reduction) is more secure against a number of attacks (side-channel analysis, software vulnerability exploitation) than executing the unmodified program in SGX, but orders of magnitude faster than executing the program using fully homomorphic encryption.

7 Conclusion

This chapter concludes our work. Section 7.1 summarizes our achievements and relates them to our primary research question. Section 7.2 discusses open problems and directions for future work.

7.1 Summary

In Chapter 1, we discussed the motivation of our work and described our scientific contributions. Section 1.2 proposed the primary research question investigated in this dissertation: "How can homomorphic encryption and trusted execution environments be combined into a practical architecture enabling efficient and secure computation on encrypted data?".

In Chapter 2, we presented foundational concepts as the basis for our contributions. We introduced common cryptographic principles (e.g., encryption schemes and game-based security definitions) used by DFAuth (Chapter 5) and provided various information theory and optimization problem definitions used by PASAPTO (Chapter 6).

In Chapter 3, we provided an overview on homomorphic encryption, TEEs and other approaches related to computation on encrypted data. We described that fully homomorphic encryption incurs high computational overhead and is structurally incapable of efficiently executing algorithms with high worst case but low average case complexity due to lack of control-flow support. We also motivated the necessity of a small trusted code base to reduce the surface for attacks allowing the protections of TEEs to be disarmed by exploiting software vulnerabilities.

In Chapter 4, we described the methodology used to investigate the primary research question. We first derived requirements from the research question and explained how our solution fulfills some of them by design. Then, we introduced our practicality and efficiency assessment methodology which includes software implementations and their evaluation on commercially available computer systems. Finally, we presented our security assessment methodology consisting of security models, security proofs and quantitative information flow techniques.

In Chapter 5, we first combined partially homomorphic encryption with a TEE to execute programs on encrypted data. By operating on control-flow driven programs rather than circuits, our approach supports the execution of efficient algorithms without incurring the complexity penalty as in FHE. By incorporating a small trusted module (implemented in a TEE), our architecture provides a small and program-independent TCB, which can be reused across applications and hardened against software vulnerabilities and side-channels. We considered control-flow leakage under active attacks, introduced the concept of dataflow authentication (DFAuth) to prevent the adversary from deviating from the dataflow of the outsourced program, and showed its interference equivalence property in a program dependency graph. We implemented DFAuth using a novel homomorphic authenticated symmetric encryption (HASE) scheme, for which we provided security definitions, two constructions and security proofs. We showed the practical feasibility of our architecture using a Java bytecode-to-bytecode compiler and evaluated transformed programs on a commercially available desktop computer. In summary, DFAuth in combination with HASE provides a practical architecture enabling efficient computations on encrypted data without the drawbacks of solutions based solely on FHE or a TEE.

Also in Chapter 5, we improved the performance of DFAuth by complementing HASE with the concept of trusted authenticated ciphertext operations (TACO). We provided updated security definitions, a generic construction and security proofs. We extended our implementation and evaluated transformed programs in a commercially available cloud environment. Our TACO construction makes use of a common authenticated symmetric encryption scheme instead of partially homomorphic encryption. Although ciphertext operations have to be performed in the trusted module, our experiments showed that TACO significantly improves the running times of outsourced programs. In summary, TACO allows DFAuth to be applied to applications which require fast response times such as the smart charging scheduler for electric vehicles considered in our evaluation.

In Chapter 6, we further improved the security of DFAuth and addressed the concern that control-flow leakage might be inacceptable from a security perspective. We formalized the problem of policy-aware security and performance trade-off (PASAPTO) analysis as an optimization problem and proved the NP-hardness of the corresponding decision problem. We presented two heuristic algorithms approximating the Pareto front of the optimization problem: a greedy heuristic providing fast convergence and a genetic algorithm providing well distributed trade-offs. We used established quantitative information flow techniques to measure security, running time to measure performance and program transformation techniques to alter the trade-off between the two. We also adjusted an existing QIF analysis to capture the adversarial information flow for each variable of a program such that we can support variable-based information flow policies. We implemented our algorithms and evaluated them on programs computing on encrypted data using DFAuth in a commercially available cloud environment. In summary, PASAPTO improves the security of DFAuth and provides developers more control over the security and performance characteristics of their outsourced program.

In conclusion, we presented a novel architecture combining cryptographic primitives with hardware-based security to compute on encrypted data. The proposed architecture has been demonstrated to be practical in various evaluations on commercially available computer systems, computations on encrypted data are efficient due to control-flow support und fast ciphertext operations, and the architecture provides extensive security guarantees.

7.2 Outlook

The architecture presented in this work addresses the problem of secure and efficient computation on encrypted data. However, we can only provide first steps towards a thorough “general purpose cloud computing on encrypted data” scenario. To bridge the gap between general purpose computing and computation on encrypted data, additional research and engineering seems necessary. In the following, we identify some of the open problems and provide some possible directions for future work.

- DFAuth considers a simple scenario between two parties, a client and a server (which has a trusted module). The cryptographic key material is generated on the trusted client and then shared with the trusted module over a secure channel. How can the DFAuth scenario be extended to cover multiple clients? Can HASE be adjusted to a public-key setting without weakening its security properties?
- Our PASAPTO analysis has been evaluated in the context of DFAuth. However, the same trade-off can be made in many other types of computation on encrypted data. In future research, the trade-off between security and performance of executing control-flow driven programs inside SGX enclaves could be investigated. PASAPTO can also be applied to MPC protocols, which may leak some intermediate results to avoid expensive MPC computations.
- The DFAuth architecture provides data confidentiality, but does not protect access patterns from being revealed to the adversary (cf. Section 3.2.2). To see that this leakage can be a security issue also in this scenario, consider a program statement of the form $a[i]$, in which some array a is accessed using some index i. Even if the content of a and the index i are encrypted, the adversary may still be able to observe the memory locations accessed by the outsourced program [RLT15]. In future work, DFAuth could be combined with ORAM (cf. Section 3.2.2) to prevent data from leaking via memory access pattern side-channels.
- The DFAuth trusted module in this work was implemented using an SGX enclave. Recently, a number of additional TEEs have become available on the market of commercial computer systems. Future work could analyze their benefits and whether the DFAuth architecture can gain any security or performance improvements from switching to another TEE.

Publications and Contributions

The concepts presented in this dissertation have been the subject of multiple scientific publications co-authored by me. In the following these publications are listed and my contributions to them are described.

- ***Andreas Fischer***, *Benny Fuhry, Florian Kerschbaum, Eric Bodden: Computation on Encrypted Data using Dataflow Authentication. In 20th Privacy Enhancing Technologies Symposium (PETS), 2020.* [Fis+20a]

  I am the principal author of this publication. Chapter 5 is based on this work. The concepts of dataflow authentication (DFAuth) and homomorphic authenticated symmetric encryption (HASE) were inspired by security issues posed by related work [Top+13]. DFAuth was developed in technical discussions with Eric Bodden and Florian Kerschbaum. HASE was developed in technical discussions with Florian Kerschbaum. Benny Fuhry helped with the Intel SGX implementation. Benny Fuhry, Florian Kerschbaum and Eric Bodden co-authored the publication.

- ***Andreas Fischer***, *Benny Fuhry, Jörn Kussmaul, Jonas Janneck, Florian Kerschbaum, Eric Bodden: Computation on Encrypted Data using Dataflow Authentication. Under submission.* [Fis+]

  I am the principal author of this publication. Chapter 5 is based on this work. The concept of trusted authenticated ciphertext operations (TACO) was developed in technical discussions with Jörn Kussmaul to improve the performance of DFAuth. Jonas Janneck helped with the implementation of the experiments. Florian Kerschbaum and Eric Bodden provided feedback. Jörn Kussmaul and Jonas Janneck co-authored the publication.

- ***Andreas Fischer***, *Jonas Janneck, Jörn Kussmaul, Nikolas Krätzschmar, Florian Kerschbaum, Eric Bodden: PASAPTO: Policy-aware Security and Performance Trade-off Analysis – Computation on Encrypted Data with Restricted Leakage. In 33rd IEEE Computer Security Foundations (CSF) Symposium, 2020.* [Fis+20b]

  I am the principal author of this publication. Chapter 6 is based on this work. The concept of policy-aware security and performance trade-off (PASAPTO) analysis was developed in technical discussions with Jonas Janneck and Florian Kerschbaum to improve the security of DFAuth further. Jörn Kussmaul helped with the implementation of the experiments. Nikolas Krätzschmar helped with the implementation of the program security measures. Florian Kerschbaum and Eric Bodden provided feedback. Jonas Janneck and Jörn Kussmaul co-authored the publication.

Bibliography

[Abr+17] I. Abraham, C. W. Fletcher, K. Nayak, B. Pinkas, and L. Ren. "Asymptotically Tight Bounds for Composing ORAM with PIR". In: *Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part I*. Ed. by S. Fehr. Vol. 10174. Lecture Notes in Computer Science. Springer, 2017, pp. 91–120 (cit. on p. 24).

[Aes] *Specification for the Advanced Encryption Standard (AES)*. Federal Information Processin Standards Publication 197. 2001. URL: `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf` (cit. on pp. 2, 60).

[AFV11] S. Agrawal, D. M. Freeman, and V. Vaikuntanathan. "Functional Encryption for Inner Product Predicates from Learning with Errors". In: *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*. Ed. by D. H. Lee and X. Wang. Vol. 7073. Lecture Notes in Computer Science. Springer, 2011, pp. 21–40 (cit. on p. 22).

[Aga00] J. Agat. "Transforming Out Timing Leaks". In: *POPL 2000, Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Boston, Massachusetts, USA, January 19-21, 2000*. Ed. by M. N. Wegman and T. W. Reps. ACM, 2000, pp. 40–53 (cit. on p. 99).

[Agr+04] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. "Order-Preserving Encryption for Numeric Data". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, June 13-18, 2004*. Ed. by G. Weikum, A. C. König, and S. Deßloch. ACM, 2004, pp. 563–574 (cit. on p. 20).

[Alm+18] J. B. Almeida, M. Barbosa, G. Barthe, H. Pacheco, V. Pereira, and B. Portela. *Enforcing ideal-world leakage bounds in real-world secret sharing MPC frameworks*. Cryptology ePrint Archive, Report 2018/404. `https://eprint.iacr.org/2018/404`. 2018 (cit. on pp. 108, 109).

[Ana+13] I. Anati, S. Gueron, S. P. Johnson, and V. R. Scarlata. "Innovative Technology for CPU Based Attestation and Sealing". In: *Workshop on Hardware and Architectural Support for Security and Privacy*. HASP. 2013 (cit. on pp. 2, 27, 28).

[Aud+18] C. Audet, J. Bigeon, D. Cartier, S. Le Digabel, and L. Salomon. "Performance indicators in multiobjective optimization". In: *Optimization Online* (2018) (cit. on p. 17).

[AWZ88] B. Alpern, M. N. Wegman, and F. K. Zadeck. "Detecting equality of variables in programs". In: *Proceedings of the 15th ACM Symposium on Principles of Programming Languages.* POPL. 1988 (cit. on p. 41).

[Ayd+18] A. Aydin, W. Eiers, L. Bang, T. Brennan, M. Gavrilov, T. Bultan, and F. Yu. "Parameterized model counting for string and numeric constraints". In: *ESEC/SIGSOFT FSE.* ACM, 2018, pp. 400–410 (cit. on p. 100).

[Bar+12] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. "On the (im)possibility of obfuscating programs". In: *J. ACM* 59.2 (2012), 6:1–6:48 (cit. on p. 25).

[Bar94] A. I. Barvinok. "A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed". In: *Mathematics of Operations Research* 19.4 (1994), pp. 769–779 (cit. on p. 100).

[BB03] D. Brumley and D. Boneh. "Remote Timing Attacks Are Practical". In: *Proceedings of the 12th USENIX Security Symposium, Washington, D.C., USA, August 4-8, 2003.* 2003 (cit. on p. 80).

[BCF17] M. Barbosa, D. Catalano, and D. Fiore. "Labeled Homomorphic Encryption - Scalable and Privacy-Preserving Processing of Outsourced Data". In: *Proceedings of the 22nd European Symposium on Research in Computer Security.* ESORICS. 2017 (cit. on p. 77).

[BDP13] P. Braione, G. Denaro, and M. Pezzè. "Enhancing symbolic execution with built-in term rewriting and constrained lazy initialization". In: *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013.* Ed. by B. Meyer, L. Baresi, and M. Mezini. ACM, 2013, pp. 411–421 (cit. on p. 100).

[BDP15] P. Braione, G. Denaro, and M. Pezzè. "Symbolic execution of programs with heap inputs". In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015.* Ed. by E. D. Nitto, M. Harman, and P. Heymans. ACM, 2015, pp. 602–613 (cit. on p. 100).

[Ber06] D. J. Bernstein. "Curve25519: New Diffie-Hellman Speed Records". In: *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings.* Ed. by M. Yung, Y. Dodis, A. Kiayias, and T. Malkin. Vol. 3958. Lecture Notes in Computer Science. Springer, 2006, pp. 207–228 (cit. on pp. 63, 80).

[BGN05] D. Boneh, E. Goh, and K. Nissim. "Evaluating 2-DNF Formulas on Ciphertexts". In: *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings.* Ed. by J. Kilian. Vol. 3378. Lecture Notes in Computer Science. Springer, 2005, pp. 325–341 (cit. on p. 23).

[BGV14] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. "(Leveled) Fully Homomorphic Encryption without Bootstrapping". In: *ACM Trans. Comput. Theory* 6.3 (2014), 13:1–13:36 (cit. on p. 23).

[BKR09] M. Backes, B. Köpf, and A. Rybalchenko. "Automatic Discovery and Quantification of Information Leaks". In: *30th IEEE Symposium on Security and Privacy (S&P 2009), 17-20 May 2009, Oakland, California, USA*. IEEE Computer Society, 2009, pp. 141–153. ISBN: 978-0-7695-3633-0 (cit. on pp. 84, 85, 102, 109).

[BN08] M. Bellare and C. Namprempre. "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm". In: *Journal of Cryptology* 21.4 (2008) (cit. on p. 77).

[Bon+04] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. "Public Key Encryption with Keyword Search". In: *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings.* Ed. by C. Cachin and J. Camenisch. Vol. 3027. Lecture Notes in Computer Science. Springer, 2004, pp. 506–522 (cit. on pp. 21, 22).

[Bon+09] D. Boneh, D. Freeman, J. Katz, and B. Waters. "Signing a linear subspace: Signature schemes for network coding". In: *Proceedings of the 12th International Workshop on Public Key Cryptography.* PKC. 2009 (cit. on p. 77).

[Bon+15] D. Boneh, K. Lewi, M. Raykova, A. Sahai, M. Zhandry, and J. Zimmerman. "Semantically Secure Order-Revealing Encryption: Multi-input Functional Encryption Without Obfuscation". In: *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II.* Ed. by E. Oswald and M. Fischlin. Vol. 9057. Lecture Notes in Computer Science. Springer, 2015, pp. 563–594 (cit. on p. 20).

[Bor+17] M. Borges, Q. Phan, A. Filieri, and C. S. Pasareanu. "Model-Counting Approaches for Nonlinear Numerical Constraints". In: *NASA Formal Methods - 9th International Symposium, NFM 2017, Moffett Field, CA, USA, May 16-18, 2017, Proceedings.* 2017, pp. 131–138 (cit. on p. 100).

[Bös+14] C. Bösch, P. H. Hartel, W. Jonker, and A. Peter. "A Survey of Provably Secure Searchable Encryption". In: *ACM Comput. Surv.* 47.2 (2014), 18:1–18:51 (cit. on p. 21).

[BR06] M. Bellare and P. Rogaway. “Code-Based Game-Playing Proofs and the Security of Triple Encryption”. In: *Proceedings of the 25th International Conference on Advances in Cryptology.* EUROCRYPT. 2006 (cit. on pp. 8, 33).

[Bra+17] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi. “Software Grand Exposure: SGX Cache Attacks Are Practical”. In: *Proceedings of the 11th USENIX Workshop on Offensive Technologies.* WOOT. 2017 (cit. on pp. 2, 28).

[BSW11] D. Boneh, A. Sahai, and B. Waters. “Functional encryption: Definitions and challenges”. In: *Proceedings of the 8th Theory of Cryptography Conference.* TCC. 2011 (cit. on p. 21).

[Bul+18] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx. “Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution”. In: *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.* Ed. by W. Enck and A. P. Felt. USENIX Association, 2018, pp. 991–1008 (cit. on p. 29).

[BV14] Z. Brakerski and V. Vaikuntanathan. “Efficient Fully Homomorphic Encryption from (Standard) LWE”. In: *SIAM J. Comput.* 43.2 (2014), pp. 831–871 (cit. on p. 23).

[Cau+17] S. Cauligi, G. Soeller, F. Brown, B. Johannesmeyer, Y. Huang, R. Jhala, and D. Stefan. “FaCT: A Flexible, Constant-Time Programming Language”. In: *IEEE Cybersecurity Development, SecDev 2017, Cambridge, MA, USA, September 24-26, 2017.* 2017, pp. 69–76 (cit. on p. 109).

[Che+19] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. Lai. “SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution”. In: *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019.* IEEE, 2019, pp. 142–157 (cit. on p. 29).

[CKN14] T. Chothia, Y. Kawamoto, and C. Novakovic. “Leakwatch: Estimating information leakage from java programs”. In: *European Symposium on Research in Computer Security.* Springer. 2014, pp. 219–236 (cit. on p. 109).

[CMP14] D. Catalano, A. Marcedone, and O. Puglisi. “Authenticating Computation on Groups: New Homomorphic Primitives and Applications”. In: *Proceedings of the 20th International Conference on the Advances in Cryptology.* ASIACRYPT. 2014 (cit. on p. 77).

[CMS05] M. R. Clarkson, A. C. Myers, and F. B. Schneider. “Belief in information flow”. In: *18th IEEE Computer Security Foundations Workshop (CSFW’05).* IEEE. 2005, pp. 31–45 (cit. on p. 109).

[CT06] T. M. Cover and J. A. Thomas. *Elements of information theory (2. ed.)* Wiley, 2006. ISBN: 978-0-471-24195-9 (cit. on p. 15).

[Dan51] G. B. Dantzig. “Maximization of a linear function of variables subject to linear inequalities”. In: *Activity analysis of production and allocation* 13 (1951), pp. 339–347 (cit. on p. 80).

[Dev+16] S. Devadas, M. van Dijk, C. W. Fletcher, L. Ren, E. Shi, and D. Wichs. “Onion ORAM: A Constant Bandwidth Blowup Oblivious RAM”. In: *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part II.* Ed. by E. Kushilevitz and T. Malkin. Vol. 9563. Lecture Notes in Computer Science. Springer, 2016, pp. 145–174 (cit. on p. 24).

[Dij+10] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. “Fully Homomorphic Encryption over the Integers”. In: *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings.* Ed. by H. Gilbert. Vol. 6110. Lecture Notes in Computer Science. Springer, 2010, pp. 24–43 (cit. on p. 23).

[DK15] G. Doychev and B. Köpf. “Rational Protection against Timing Attacks”. In: *IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015.* Ed. by C. Fournet, M. W. Hicks, and L. Viganò. IEEE Computer Society, 2015, pp. 526–536 (cit. on p. 108).

[DL+04] J. A. De Loera, R. Hemmecke, J. Tauzer, and R. Yoshida. “Effective lattice point counting in rational convex polytopes”. In: *Journal of symbolic computation* 38.4 (2004), pp. 1273–1302 (cit. on p. 100).

[DMD16] Y. Dong, A. Milanova, and J. Dolby. “JCrypt: Towards Computation over Encrypted Data”. In: *Proceedings of the 13th International Conference on Principles and Practices of Programming on the Java Platform.* PPPJ. 2016 (cit. on pp. 26, 36, 41, 78, 100).

[EH06] D. Eastlake 3rd and T. Hansen. *US Secure Hash Algorithms (SHA and HMAC-SHA).* RFC 4634 (Informational). Internet Engineering Task Force, 2006 (cit. on p. 60).

[EKR18] D. Evans, V. Kolesnikov, and M. Rosulek. “A Pragmatic Introduction to Secure Multi-Party Computation”. In: *Found. Trends Priv. Secur.* 2.2-3 (2018), pp. 70–246 (cit. on p. 24).

[Elg85] T. Elgamal. “A public key cryptosystem and a signature scheme based on discrete logarithms”. In: *IEEE Transactions on Information Theory* 31.4 (1985) (cit. on p. 11).

[FF+93] C. M. Fonseca, P. J. Fleming, et al. “Genetic Algorithms for Multiobjective Optimization: FormulationDiscussion and Generalization.” In: *Icga.* Vol. 93. July. 1993, pp. 416–423 (cit. on pp. 94, 95).

[FGS19] O. Frendo, N. Gaertner, and H. Stuckenschmidt. "Real-time smart charging based on precomputed schedules". In: *IEEE Transactions on Smart Grid* (2019) (cit. on pp. 71–73).

[Fis+] A. Fischer, B. Fuhry, J. Kussmaul, J. Janneck, F. Kerschbaum, and E. Bodden. "Computation on Encrypted Data using Dataflow Authentication". Under submission (cit. on pp. 4, 35, 115).

[Fis+17] A. Fischer, B. Fuhry, F. Kerschbaum, and E. Bodden. "Computation on Encrypted Data using Data Flow Authentication". In: *CoRR* abs/1710.00390 (2017). arXiv: `1710.00390`. URL: `http://arxiv.org/abs/1710.00390` (cit. on p. 3).

[Fis+20a] A. Fischer, B. Fuhry, F. Kerschbaum, and E. Bodden. "Computation on Encrypted Data using Dataflow Authentication". In: *20th Privacy Enhancing Technologies Symposium (PETS)* 2020.1 (2020). URL: `https://petsymposium.org/2020/files/papers/issue1/popets-2020-0002.pdf` (cit. on pp. 3, 35, 115).

[Fis+20b] A. Fischer, J. Janneck, J. Kussmaul, N. Krätzschmar, F. Kerschbaum, and E. Bodden. "PASAPTO: Policy-aware Security and Performance Trade-off Analysis – Computation on Encrypted Data with Restricted Leakage". In: *33rd IEEE Computer Security Foundations Symposium, CSF 2020, Boston, MA, USA, June 22-26, 2020.* IEEE, 2020, pp. 230–245 (cit. on pp. 4, 79, 115).

[FKN94] U. Feige, J. Kilian, and M. Naor. "A minimal model for secure computation (extended abstract)". In: *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada.* Ed. by F. T. Leighton and M. T. Goodrich. ACM, 1994, pp. 554–563 (cit. on p. 24).

[Gar+13] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. "Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits". In: *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*. IEEE Computer Society, 2013, pp. 40–49 (cit. on p. 25).

[Gar+16] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. "Candidate Indistinguishability Obfuscation and Functional Encryption for All Circuits". In: *SIAM J. Comput.* 45.3 (2016), pp. 882–929 (cit. on p. 21).

[Gen09] C. Gentry. "Fully homomorphic encryption using ideal lattices". In: *Proceedings of the Symposium on Theory of Computing.* STOC. 2009 (cit. on pp. 2, 23).

[GGP11] R. Gennaro, C. Gentry, and B. Parno. "Non-interactive verifiable computing: Outsourcing computation to untrusted workers". In: *Proceedings of the 30th International Conference on Advances in Cryptology.* CRYPTO. 2011 (cit. on p. 78).

[GHS12] C. Gentry, S. Halevi, and N. P. Smart. “Homomorphic evaluation of the AES circuit”. In: *Proceedings of the 32nd International Conference on Advances in Cryptology*. CRYPTO. 2012 (cit. on pp. 2, 23).

[Gil+16] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing. “CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy”. In: *Proceedings of the 33rd International Conference on Machine Learning*. ICML. 2016 (cit. on pp. 70, 71).

[GM82] S. Goldwasser and S. Micali. “Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information”. In: *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*. Ed. by H. R. Lewis, B. B. Simons, W. A. Burkhard, and L. H. Landweber. ACM, 1982, pp. 365–377 (cit. on p. 23).

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. “How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority”. In: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*. Ed. by A. V. Aho. ACM, 1987, pp. 218–229 (cit. on p. 24).

[Gol+13a] S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. “How to Run Turing Machines on Encrypted Data”. In: *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*. Ed. by R. Canetti and J. A. Garay. Vol. 8043. Lecture Notes in Computer Science. Springer, 2013, pp. 536–553 (cit. on p. 21).

[Gol+13b] S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. “Reusable garbled circuits and succinct functional encryption”. In: *Proceedings of the Symposium on Theory of Computing*. STOC. 2013 (cit. on p. 21).

[Göt+17] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller. “Cache Attacks on Intel SGX”. In: *Proceedings of the 10th European Workshop on Systems Security, EUROSEC 2017, Belgrade, Serbia, April 23, 2017*. Ed. by C. Giuffrida and A. Stavrou. ACM, 2017, 2:1–2:6 (cit. on p. 28).

[GR+87] D. E. Goldberg, J. Richardson, et al. “Genetic algorithms with sharing for multimodal function optimization”. In: *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*. Hillsdale, NJ: Lawrence Erlbaum. 1987, pp. 41–49 (cit. on pp. 94, 95).

[GW13] R. Gennaro and D. Wichs. "Fully Homomorphic Message Authenticators". In: *Proceedings of the 19th International Conference on the Advances in Cryptology.* ASIACRYPT. 2013 (cit. on p. 77).

[Hac+02] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra. "Executing SQL over encrypted data in the database-service-provider model". In: *Proceedings of the ACM International Conference on Management of Data.* SIGMOD. 2002 (cit. on p. 26).

[Har08] D. Harkins. "Synthetic Initialization Vector (SIV) Authenticated Encryption Using the Advanced Encryption Standard (AES)". In: *RFC* 5297 (2008), pp. 1–26 (cit. on p. 20).

[HM10] J. Heusser and P. Malacaria. "Quantifying information leak vulnerabilities". In: *arXiv preprint arXiv:1007.0918* (2010) (cit. on p. 109).

[HMS12] Y. Hu, W. Martin, and B. Sunar. "Enhanced Flexibility for Homomorphic Encryption Schemes via CRT". In: *Proceedings (Industrial Track) of the 10th International Conference on Applied Cryptography and Network Security.* ACNS. 2012 (cit. on pp. 47, 66).

[Hoe+13] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo. "Using Innovative Instructions to Create Trustworthy Software Solutions". In: *Workshop on Hardware and Architectural Support for Security and Privacy.* HASP. 2013 (cit. on pp. 2, 27).

[Hol+92] J. H. Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* MIT press, 1992 (cit. on p. 94).

[Int19] Intel Corporation. *Intel Software Guard Extensions (Intel SGX) SDK for Linux OS.* 2019. URL: `https://download.01.org/intel-sgx/linux-2.6/docs/Intel_SGX_Developer_Reference_Linux_2.6_Open_Source.pdf` (visited on 10/12/2020) (cit. on p. 29).

[JY14] C. Joo and A. Yun. "Homomorphic Authenticated Encryption Secure against Chosen-Ciphertext Attack". In: *Proceedings of the 20th International Conf. on the Advances in Cryptology.* ASIACRYPT. 2014 (cit. on p. 77).

[KB07] B. Köpf and D. Basin. "An information-theoretic model for adaptive side-channel attacks". In: *Proceedings of the 14th ACM conference on Computer and communications security.* ACM. 2007, pp. 286–296 (cit. on p. 109).

[Kin76] J. C. King. "Symbolic execution and program testing". In: *Communications of the ACM* 19.7 (1976), pp. 385–394 (cit. on p. 84).

[KK03] T. Kivinen and M. Kojo. *More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE).* RFC 3526 (Proposed Standard). Internet Engineering Task Force, 2003 (cit. on p. 63).

[KL08] J. Katz and Y. Lindell. “Aggregate message authentication codes”. In: *Proceedings of the Cryptographers’ Track of the RSA Conference.* CT-RSA. 2008 (cit. on p. 77).

[KL14] J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition.* 2nd. Chapman & Hall/CRC, 2014. ISBN: 1466570261, 9781466570269 (cit. on pp. 8, 22).

[Kle14] V. Klebanov. “Precise quantitative information flow analysis - a symbolic approach”. In: *Theor. Comput. Sci.* 538 (2014), pp. 124–139 (cit. on pp. 84, 109).

[KM70] V. Klee and G. J. Minty. *How good is the simplex algorithm.* Tech. rep. University of Washington, Seattle Department of Mathematics, 1970 (cit. on p. 80).

[KO97] E. Kushilevitz and R. Ostrovsky. “Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval”. In: *38th Annual Symposium on Foundations of Computer Science, FOCS ’97, Miami Beach, Florida, USA, October 19-22, 1997.* IEEE Computer Society, 1997, pp. 364–373 (cit. on p. 25).

[Koc+19] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. “Spectre Attacks: Exploiting Speculative Execution”. In: *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019.* IEEE, 2019, pp. 1–19 (cit. on p. 29).

[Kuč+17] M. Kučera, P. Tsankov, T. Gehr, M. Guarnieri, and M. Vechev. “Synthesis of probabilistic privacy enforcement”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.* ACM. 2017, pp. 391–408 (cit. on p. 109).

[Lam+11] P. Lam, E. Bodden, O. Lhotak, and L. Hendren. “The Soot framework for Java program analysis: a retrospective”. In: *Cetus Users and Compiler Infastructure Workshop.* CETUS. 2011 (cit. on pp. 59, 100).

[Lee+17a] J. Lee, J. Jang, Y. Jang, N. Kwak, Y. Choi, C. Choi, T. Kim, M. Peinado, and B. B. Kang. “Hacking in Darkness: Return-oriented Programming against Secure Enclaves”. In: *Proceedings of the 26th USENIX Security Symposium.* USENIX Security. 2017 (cit. on pp. 2, 29, 35).

[Lee+17b] S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado. “Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing”. In: *Proceedings of the 26th USENIX Security Symposium.* USENIX Security. 2017 (cit. on pp. 2, 28, 36, 79).

[Lew+10] A. B. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters. “Fully Secure Functional Encryption: Attribute-Based Encryption and (Hierarchical) Inner Product Encryption”. In: *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings.* Ed. by H. Gilbert. Vol. 6110. Lecture Notes in Computer Science. Springer, 2010, pp. 62–91 (cit. on p. 22).

[LHS13] C. Liu, M. Hicks, and E. Shi. “Memory trace oblivious program execution”. In: *2013 IEEE 26th Computer Security Foundations Symposium.* IEEE. 2013, pp. 51–65 (cit. on p. 108).

[Lib] *The Sodium crypto library (libsodium).* URL: `https://download.libsodium.org/doc/` (visited on 10/12/2020) (cit. on pp. 60, 63).

[Lip+18] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg. “Meltdown: Reading Kernel Memory from User Space”. In: *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.* Ed. by W. Enck and A. P. Felt. USENIX Association, 2018, pp. 973–990 (cit. on p. 29).

[Liu+15] C. Liu, A. Harris, M. Maas, M. W. Hicks, M. Tiwari, and E. Shi. “GhostRider: A Hardware-Software System for Memory Trace Oblivious Computation”. In: *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems.* ASPLOS. 2015 (cit. on pp. 76, 109).

[LO05] J. Li and E. R. Omiecinski. “Efficiency and security trade-off in supporting range queries on encrypted databases”. In: *IFIP Annual Conference on Data and Applications Security and Privacy.* Springer. 2005, pp. 69–83 (cit. on p. 108).

[Mah95] S. W. Mahfoud. “Niching methods for genetic algorithms”. PhD thesis. 1995 (cit. on p. 94).

[Mal11] P. Malacaria. “Algebraic Foundations for Information Theoretical, Probabilistic and Guessability measures of Information Flow”. In: *CoRR* abs/1101.3453 (2011). URL: `http://arxiv.org/abs/1101.3453` (cit. on pp. 84, 109).

[Mal+18] P. Malacaria, M. Khouzani, C. S. Pasareanu, Q.-S. Phan, and K. Luckow. “Symbolic side-channel analysis for probabilistic programs”. In: *2018 IEEE 31st Computer Security Foundations Symposium (CSF).* IEEE. 2018, pp. 313–327 (cit. on p. 109).

[McK+13] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. “Innovative Instructions and Software Model for Isolated Execution”. In: *Workshop on Hardware and Architectural Support for Security and Privacy.* HASP. 2013 (cit. on pp. 2, 27).

[MIE17] A. Moghimi, G. Irazoqui, and T. Eisenbarth. “CacheZoom: How SGX Amplifies the Power of Cache Attacks”. In: *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings.* Ed. by W. Fischer and N. Homma. Vol. 10529. Lecture Notes in Computer Science. Springer, 2017, pp. 69–90 (cit. on p. 28).

[Mie98] K. Miettinen. *Nonlinear multiobjective optimization.* Vol. 12. International series in operations research and management science. Kluwer, 1998. ISBN: 978-0-7923-8278-2 (cit. on p. 16).

[Mol+05] D. Molnar, M. Piotrowski, D. Schultz, and D. A. Wagner. “The Program Counter Security Model: Automatic Detection and Removal of Control-Flow Side Channel Attacks”. In: *Information Security and Cryptology - ICISC 2005, 8th International Conference, Seoul, Korea, December 1-2, 2005, Revised Selected Papers.* ICISC. 2005 (cit. on pp. 76, 99, 108, 109).

[Mpi] *MPIR: Multiple Precision Integers and Rationals.* URL: `http://mpir.org` (visited on 10/12/2020) (cit. on p. 60).

[Nay+17] K. Nayak, C. W. Fletcher, L. Ren, N. Chandran, S. V. Lokam, E. Shi, and V. Goyal. “HOP: Hardware makes Obfuscation Practical”. In: *24th Annual Network and Distributed System Security Symposium.* NDSS. 2017 (cit. on p. 77).

[Net] *AT&T Global IP Network – Network Averages.* URL: `http://ipnetwork.bgtmo.ip.att.net/pws/averages.html` (visited on 09/08/2017) (cit. on p. 65).

[Neu] *Neuroph – Java Neural Network Framework.* URL: `http://neuroph.sourceforge.net` (visited on 10/12/2020) (cit. on p. 67).

[NKP18] Y. Noller, R. Kersten, and C. S. Păsăreanu. “Badger: Complexity analysis with fuzzing and symbolic execution”. In: *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis.* ACM. 2018, pp. 322–332 (cit. on p. 109).

[Ohr+16] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa. “Oblivious Multi-Party Machine Learning on Trusted Processors”. In: *Proceedings of the 25th USENIX Security Symposium.* USENIX Security. 2016 (cit. on pp. 70, 71).

[OT11] J. Obdržálek and M. Trtík. “Efficient loop navigation for symbolic execution”. In: *International Symposium on Automated Technology for Verification and Analysis.* Springer. 2011, pp. 453–462 (cit. on p. 109).

[Pai99a] P. Paillier. “Public-key Cryptosystems Based on Composite Degree Residuosity Classes”. In: *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques.* EUROCRYPT. 1999 (cit. on p. 77).

[Pai99b] P. Paillier. “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes”. In: *Advances in Cryptology - EUROCRYPT ’99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding.* Ed. by J. Stern. Vol. 1592. Lecture Notes in Computer Science. Springer, 1999, pp. 223–238 (cit. on p. 23).

[PM14] Q.-S. Phan and P. Malacaria. “Abstract model counting: a novel approach for quantification of information leaks”. In: *Proceedings of the 9th ACM symposium on Information, computer and communications security.* ACM. 2014, pp. 283–292 (cit. on p. 109).

[Pop+11] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. “CryptDB: protecting confidentiality with encrypted query processing”. In: *Proceedings of the 23rd ACM Symposium on Operating Systems Principles.* SOSP. 2011 (cit. on p. 26).

[PR12] O. Pandey and Y. Rouselakis. “Property Preserving Symmetric Encryption”. In: *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings.* Ed. by D. Pointcheval and T. Johansson. Vol. 7237. Lecture Notes in Computer Science. Springer, 2012, pp. 375–391 (cit. on p. 19).

[RAD78] R. L. Rivest, L Adleman, and M. L. Dertouzos. “On Data Banks and Privacy Homomorphisms”. In: *Foundations of Secure Computation, Academia Press* (1978), pp. 169–179 (cit. on p. 2).

[Ren+15] L. Ren, C. W. Fletcher, A. Kwon, E. Stefanov, E. Shi, M. van Dijk, and S. Devadas. “Constants Count: Practical Improvements to Oblivious RAM”. In: *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015.* Ed. by J. Jung and T. Holz. USENIX Association, 2015, pp. 415–430 (cit. on p. 24).

[RLT15] A. Rane, C. Lin, and M. Tiwari. “Raccoon: Closing Digital Side-Channels through Obfuscated Execution”. In: *Proceedings of the 24th USENIX Security Symposium.* USENIX Security. 2015 (cit. on pp. 77, 99, 109, 113).

[RSA78] R. L. Rivest, A. Shamir, and L. M. Adleman. “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”. In: *Commun. ACM* 21.2 (1978), pp. 120–126 (cit. on p. 2).

[RVLB15] N. Riquelme, C. Von Lücken, and B. Baran. “Performance metrics in multi-objective optimization”. In: *2015 Latin American Computing Conference (CLEI).* IEEE. 2015, pp. 1–11 (cit. on p. 17).

[Sch+] S. van Schaik, A. Kwong, D. Genkin, and Y. Yarom. *SGAxe: How SGX Fails in Practice.* URL: `https://sgaxeattack.com/` (visited on 10/12/2020) (cit. on p. 29).

[Sch+17] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard. “Malware Guard Extension: Using SGX to Conceal Cache Attacks”. In: *Proceedings of the 14th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment.* DIMVA. 2017 (cit. on pp. 2, 28).

[Sch+19] M. Schwarz, M. Lipp, D. Moghimi, J. V. Bulck, J. Stecklina, T. Prescher, and D. Gruss. “ZombieLoad: Cross-Privilege-Boundary Data Sampling”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019.* Ed. by L. Cavallaro, J. Kinder, X. Wang, and J. Katz. ACM, 2019, pp. 753–768 (cit. on p. 29).

[Sch+20] S. van Schaik, M. Minkin, A. Kwong, D. Genkin, and Y. Yarom. “CacheOut: Leaking Data on Intel CPUs via Cache Evictions”. In: *CoRR* abs/2006.13353 (2020). arXiv: `2006.13353`. URL: `https://arxiv.org/abs/2006.13353` (cit. on p. 29).

[SM03] A. Sabelfeld and A. C. Myers. “Language-based information-flow security”. In: *IEEE Journal on selected areas in communications* 21.1 (2003), pp. 5–19 (cit. on p. 108).

[Sma+14] N. P. Smart, V. Rijmen, B. Gierlichs, K. G. Paterson, M. Stam, B. Warinschi, and G. Watson. *Algorithms, key size and parameters report.* Ed. by N. P. Smart. 2014 (cit. on p. 47).

[Smi07] G. Smith. “Principles of Secure Information Flow Analysis”. In: *Malware Detection.* Ed. by M. Christodorescu, S. Jha, D. Maughan, D. Song, and C. Wang. Vol. 27. Advances in Information Security. Springer, 2007, pp. 291–307. ISBN: 978-0-387-32720-4 (cit. on p. 44).

[Smi09] G. Smith. “On the foundations of quantitative information flow”. In: *International Conference on Foundations of Software Science and Computational Structures.* Springer. 2009, pp. 288–302 (cit. on pp. 84, 109).

[Ste+13] E. Stefanov, M. van Dijk, E. Shi, C. W. Fletcher, L. Ren, X. Yu, and S. Devadas. “Path ORAM: an extremely simple oblivious RAM protocol”. In: *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS’13, Berlin, Germany, November 4-8, 2013.* Ed. by A. Sadeghi, V. D. Gligor, and M. Yung. ACM, 2013, pp. 299–310 (cit. on p. 24).

[SV10] N. P. Smart and F. Vercauteren. “Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes”. In: *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, May 26-28, 2010. Proceedings.* Ed. by P. Q. Nguyen and D. Pointcheval. Vol. 6056. Lecture Notes in Computer Science. Springer, 2010, pp. 420–443 (cit. on p. 23).

[SVJ14] R. Sumbaly, N Vishnusri, and S Jeyalatha. “Diagnosis of breast cancer using decision tree data mining technique”. In: *International Journal of Computer Applications* 98.10 (2014) (cit. on p. 104).

[SWP00] D. X. Song, D. A. Wagner, and A. Perrig. “Practical Techniques for Searches on Encrypted Data”. In: *2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14-17, 2000.* IEEE Computer Society, 2000, pp. 44–55 (cit. on pp. 20, 21).

[Tet+13] S. Tetali, M. Lesani, R. Majumdar, and T. Millstein. “MrCrypt: Static Analysis for Secure Cloud Computations”. In: *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages & Applications.* OOPSLA. 2013 (cit. on pp. 25, 36, 77).

[Top+13] S. Tople, S. Shinde, Z. Chen, and P. Saxena. “AUTOCRYPT: Enabling Homomorphic Computation on Servers to Protect Sensitive Web Content”. In: *Proceedings of the ACM International Conference on Computer & Communications Security.* CCS. 2013 (cit. on pp. 26, 36, 41, 78, 115).

[Wei+13] S. Wei, J. Wang, R. Yin, and J. Yuan. “Trade-off between security and performance in block ciphered systems with erroneous ciphertexts”. In: *IEEE Transactions on Information Forensics and Security* 8.4 (2013), pp. 636–645 (cit. on p. 108).

[Wei+16] N. Weichbrodt, A. Kurmus, P. R. Pietzuch, and R. Kapitza. “AsyncShock: Exploiting Synchronisation Bugs in Intel SGX Enclaves”. In: *Computer Security - ESORICS 2016 - 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part I.* Ed. by I. G. Askoxylakis, S. Ioannidis, S. K. Katsikas, and C. A. Meadows. Vol. 9878. Lecture Notes in Computer Science. Springer, 2016, pp. 440–457 (cit. on p. 29).

[WLS09] D. Wasserrab, D. Lohner, and G. Snelting. “On PDG-based noninterference and its modular proof”. In: *Proceedings of the 2009 Workshop on Programming Languages and Analysis for Security.* PLAS. 2009 (cit. on p. 45).

[WR10] K. Wolter and P. Reinecke. “Performance and security tradeoff”. In: *International School on Formal Methods for the Design of Computer, Communication and Software Systems.* Springer. 2010, pp. 135–167 (cit. on p. 108).

[XCP15] Y. Xu, W. Cui, and M. Peinado. “Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems”. In: *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015.* IEEE Computer Society, 2015, pp. 640–656 (cit. on p. 28).

[Yao82] A. C. Yao. "Protocols for Secure Computations (Extended Abstract)". In: *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*. IEEE Computer Society, 1982, pp. 160–164 (cit. on p. 24).

[YT11] H. Yasuoka and T. Terauchi. "On bounding problems of quantitative information flow". In: *Journal of Computer Security* 19.6 (2011), pp. 1029–1082 (cit. on p. 109).

[ZBT06] E. Zitzler, D. Brockhoff, and L. Thiele. "The Hypervolume Indicator Revisited: On the Design of Pareto-compliant Indicators Via Weighted Integration". In: *Evolutionary Multi-Criterion Optimization, 4th International Conference, EMO 2007, Matsushima, Japan, March 5-8, 2007, Proceedings*. Ed. by S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, and T. Murata. Vol. 4403. Lecture Notes in Computer Science. Springer, 2006, pp. 862–876 (cit. on pp. 16, 17).

[ZC11] W. Zeng and M.-Y. Chow. "A trade-off model for performance and security in secured networked control systems". In: *2011 IEEE International Symposium on Industrial Electronics*. IEEE. 2011, pp. 1997–2002 (cit. on p. 108).

[Zha+16] H. Zhang, Y. Shu, P. Cheng, and J. Chen. "Privacy and performance trade-off in cyber-physical systems". In: *IEEE Network* 30.2 (2016), pp. 62–66 (cit. on p. 108).

[ZT98] E. Zitzler and L. Thiele. "Multiobjective optimization using evolutionary algorithms—a comparative case study". In: *International conference on parallel problem solving from nature*. Springer. 1998, pp. 292–301 (cit. on p. 17).