Practical Integrity Protection for Private Computations

Master's Thesis

Christian Knabenhans

October 17, 2022

Supervised by Professor Kenneth Paterson, Dr Anwar Hithnawi, and Alexander Viand

Applied Cryptography Group | Privacy Preserving Systems Lab
Institute of Information Security
Department of Computer Science
ETH Zurich

Abstract

Fully Homomorphic Encryption (FHE) allows for computations to be carried out on encrypted data, and is slowly becoming used in real-world deployments (e.g., in the Microsoft Edge Password Monitor), notably thanks to theoretical improvements, implementation and hardware optimizations, as well as an ongoing ISO standardization.

Despite its tremendous potential as a building block for privacy-preserving applications, FHE does not provide computation integrity, due to its inherent malleability; when using FHE to delegate computations on encrypted data to a third party, a client does not have any guarantee that the intended computation was carried out on its data. This lack of integrity has obvious implications for the correctness of the result, but can also lead to a complete loss of privacy, as a malicious server could misuse the malleability of FHE to carry out a key-recovery attack.

While this issue has been raised in the community, it has received scant attention, as FHE only recently matured enough to be deployed. The variety of efforts studying this and related issues have mostly remained isolated efforts, split across smaller sub-communities studying only certain aspects of the overall problem.

In this thesis, we are the first to consider this issue holistically: we map and unify the space of existing approaches, analyse and evaluate their relevancy to the needs of real-world FHE deployments, and use this newly gained understanding to point out gaps and generic attacks for a wide range of FHE integrity constructions in realistic settings.

We then propose novel integrity notions for real-world FHE deployments that prevent these attacks, analyse their relation to existing notions and propose generic constructions to achieve them. In addition, we explore, improve and implement two families of promising, concrete integrity primitives, and sketch an initial design for a novel integrity scheme that would be capable of fully supporting the needs of real-world FHE deployments.

Acknowledgements

I would firstly like to express my deep gratitude to my supervisors Alexander Viand and Dr Anwar Hithnawi for their fantastic guidance and advice throughout this thesis, as well as for their generous support, which enabled me to present my ongoing research to various experts.

I also want to express my sincere thanks to Professor Kenny Paterson for his supervision of my thesis, and for the opportunity to write it in the Applied Cryptography Group.

I am grateful for the insights and comments of Erin Hales from Royal Holloway University of London, Nojan Sheybani from the University of California San Diego and Intel Labs, and of Dr Michael Steiner from the Intel Labs Crypto Frontiers Research Center. I am also grateful to Sylvain Chatel and the Laboratory for Data Security at EPFL for sparking my interest in FHE and its integrity woes in the summer before this thesis was written.

Finally, I would like to thank the members of the Applied Cryptography Group and the Privacy Preserving Systems Lab at ETH for sharing their insights, fostering a great working atmosphere, and for their knack for asking the hard questions.

Contents

Chapter 1

Introduction

Fully Homomorphic Encryption (FHE) allows for computations to be carried out on encrypted data, and is now seeing use in real-world deployments (e.g., in the Microsoft Edge Password Monitor [Lau+21]), notably thanks to theoretical improvements, implementation and hardware optimizations, as well as an ongoing ISO standardization. Despite its tremendous potential as a building block for privacy-preserving applications, FHE does not provide computation integrity, due to its inherent malleability; when using FHE to delegate computations on encrypted data to a third party, a client does not have any guarantee that the intended computation was carried out on its data.

This lack of integrity has obvious implications for the correctness of the result, but can also lead to a complete loss of privacy, as a malicious server could misuse the malleability of FHE to carry out a key-recovery attack [FHR21]. This issue has been raised in the community [ZPS12b; CT15; CGG16], but it has received scant attention, as FHE only recently matured enough to be deployed. Nevertheless, there have been a variety of efforts to study this and related issues, but these have mostly remained isolated efforts, split across smaller sub-communities studying only certain aspects of the overall problem. A variety of issues have given rise to the current complex landscape of FHE integrity.

Security Assumptions. Historically, FHE has been such a novel and non-practical concept that its security against strong adversaries was not a central tenet of FHE research. Accordingly, the FHE research community has extensively made use of the assumption that the server running an FHE application would be honest-but-curious, but not actively malicious [FV12; BGV14; Chi+20; DM15]. Practitioners outside the research community are usually not aware of this caveat, and the community (especially commercial entities) understandably does not actively advertise this limitation either.

This honest-but-curious assumption may be reasonable in some deployment scenarios (e.g., when FHE is used only to ensure regulatory compliance, when dealing with trusted or well-respected institutions cooperating on restricted data, or in the blockchain setting, where all parties perform the same computation). However, the necessity to trust the server to this extent is very limiting to the scope of application scenarios, preventing the use of any cloud computing platform that is not fully trusted. This is not possible in many settings (e.g., for the defence and intelligence sectors). In addition, this assumption does not cover the case of a compromise of an otherwise trusted party by a malicious entity. In order to broaden the applicability of FHE, FHE applications need to be strengthened against stronger adversaries and existing attacks.

Attacks and Defences. In addition to this widespread weak assumption, the security guarantees of FHE against stronger adversaries are very brittle. Indeed, FHE research has mostly focused on designing and improving FHE schemes in order to make them more practical, while the implications of using FHE as a primitive in larger protocols and applications has received much less attention. Nevertheless, some works investigated these issues, outlining both attacks and countermeasures for existing FHE schemes.

Attacks against the security of FHE have been known since shortly after the birth of FHE [ZPS12b; CGG16], notably highlighting the inherent risk of inadvertent decryption oracles in realistic FHE applications, and the resulting catastrophic consequences for confidentiality. Practical key-recovery attacks have also been developed for all major FHE schemes [ZPS12a; CT15; FHR21].

In order to remediate these attacks, a line of research has emerged that constructs more robust FHE schemes [Bon+07; Lof+12; LGM16; Lai+16; Emu+18; WWX18; Emu21; SET22], achieving indistinguishability against chosen ciphertext attacks. Unfortunately, these constructions remain very theoretical, being inefficient in practice and/or relying on extremely strong cryptographic primitives. A different line of research tries to achieve Verifiable Computation (VC) for FHE; this notion guarantees that a certain function was executed on the ciphertext, while preserving the confidentiality of inputs [GGP10; GW13; CF13; FGP14; FNP20; Boi+21; GNSV21; Cha+22]. While these VC schemes are more concretely efficient than the constructions mentioned above, there is still a significant gap between the assumptions made by existing VC schemes and the way state-of-the-art FHE schemes are used in practice. In particular, all these VC schemes can only tolerate unrealistic malicious adversaries limited to weaker oracles than what real-world settings provide.

Contributions. In this thesis, we are (to the best of our knowledge) the first to consider this issue holistically: we map the space of existing approaches, analyse and evaluate their relevancy to the needs of real-world FHE deployments, and use these insights to propose a more unified perspective on integrity for FHE. We then propose novel integrity notions for real-world FHE deployments, analyse their relation to existing notions and propose generic constructions to achieve them. In addition, we sketch an initial design for a novel integrity scheme capable of fully supporting the needs of real-world FHE deployments.

We first study, unify, and compare existing integrity protection approaches in Chapter 3. In Chapter 4, we then study two families of promising integrity primitives, improve their efficiency, and provide the first open-source implementation of two concrete instantiations. We then point out fundamental shortcomings in existing FHE integrity approaches, and show in Chapter 5 that for virtually all FHE applications, there is a significant risk of total loss of privacy and correctness in the malicious setting. To address this, we analyse the main FHE use cases, from which we extract properties that are desirable for FHE applications to inform novel, more realistic security notions. In order to further our understanding of the theoretical underpinnings of FHE integrity, we investigate existing notions from the VC literature and more traditional indistinguishability-based notions for FHE in Chapter 6. In Chapter 7, we provide generic constructions to realise our novel notions against strong adversaries in a realistic deployment setting. Finally, in Chapter 8 we sketch ideas for a new integrity scheme that is fully compatible with modern FHE and satisfies our stronger security notions.

Chapter 2

Preliminaries

We now introduce our notation and conventions, as well as the relevant background for this thesis.

2.1 Notation and Conventions

We denote by $a := b$ the assignment of b to the a; we let $a \leftarrow \chi$ denote random sampling from a distribution (and $a \leftarrow \mathcal{A}$ sampling from a probabilistic Turing machine), and let $a \leftarrow S$ denote sampling from the set S uniformly at random.

We will also use the Iverson bracket, defined as $[b] := 1$ if b is true and 0 otherwise.

When investigating the concrete efficiency of computations, concrete complexities will be defined in terms of operations on certain input spaces. We will write $n\ {}_X\oplus_Y$ to denote that a computation requires n applications of the operator $\oplus : X \times Y \to Z$.

Following the convention in the FHE literature, we identify $\mathbb{Z}_p$ with the zero-centered set $\mathbb{Z} \cap \left[-\frac{p}{2}, \frac{p}{2}\right)$, we write $[\cdot]_p$ for the modular reduction in this set, and $\lfloor \cdot \rceil$ for rounding to the nearest integer.

Let $\kappa \in \mathbb{N}$ be the security parameter throughout. We say that a function f is negligible in κ and denote it by $\mathsf{negl}(\kappa)$ if for all $c > 0$, $f(\kappa) = o(\lambda^{-c})$ For an algorithm $\mathcal{A}$, $\mathcal{A}^F(x) \Rightarrow y$ denotes that $\mathcal{A}$ outputs y (potentially non-deterministically) when run on input x, and given oracle access to $\mathcal{O}_F$. For split adversaries $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$, we implicitly allow the adversary to retain state between the invocation of $\mathcal{A}_i$ and $\mathcal{A}_{i+1}$, unless specified otherwise.

2.2 Fully Homomorphic Encryption (FHE)

Fully Homomorphic Encryption (FHE) allows arbitrary computations to be performed on encrypted data, enabling a wide range of privacy-preserving applications (privacy-preserving Machine Learning, Private Set Intersection, Private Information Retrieval, etc.) in a plethora of domains (health, banking, defence, etc.). FHE can be achieved in two ways: either by leveraging a Leveled Homomorphic Encryption (LHE) scheme (which only support computations up to some multiplicative depth), and choosing appropriate parameters for each circuit; or by using bootstrapping, which takes a ciphertext and generates an equivalent, fresh re-encryption of the underlying plaintext by homomorphically decrypting and re-encrypting. Popular FHE schemes include B/FV [FV12], BGV [BGV14], CKKS [Che+17], TFHE [Chi+20], and FHEW [DM15].

Contrary to other privacy-enhancing technologies (e.g., Multi-Party Computation (MPC)), FHE is round-efficient and relatively communication-efficient, as both the evaluation and decryption phases of a typical client-server protocol can be performed without interaction.

In recent years, FHE has become more and more practical and easy to use, owing to theoretical efficiency improvements, better software implementation [AB+22; Sea; MBo20], hardware acceleration [Boe+21], and dedicated tooling ecosystems [VJH21]. In parallel, FHE is currently being standardized by ISO [Iso].

However, as Homomorphic Encryption (HE) schemes are malleable by construction (operations on ciphertext directly relate to operations on plaintexts), they do not offer any integrity guarantees: a user has no way to check that a HE result was computed using a given circuit.

Formally, we define FHE schemes as follows:

Definition 2.1 (FHE scheme)
An FHE scheme is a tuple of PPT algorithms $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$, *where:*

- $\mathsf{KGen}(1^\kappa)$ *outputs a public key* pk, *an evaluation key* ek, *and a secret key* sk *for the security level* κ
- $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{m}; r)$ *encrypts a plaintext* $\mathbf{m}$ *to a ciphertext* $\mathbf{ct}$, *using the randomness* r. *We usually do not write out the randomness explicitly, and write* $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{m})$ *instead*
- $\mathsf{Eval}_{\mathsf{ek}}(F, C_{in}, M_{in})$ *returns a ciphertext* $\mathbf{ct}$, *corresponding to the evaluation of the FHE circuit* F *on input ciphertexts* $C_{in} = (\mathbf{ct}_1, \ldots, \mathbf{ct}_m)$ *and input plaintexts* $M_{in} = (\mathbf{m}_1, \ldots, \mathbf{m}_n)$
- $\mathsf{Dec}_{\mathsf{sk}}(\mathbf{ct})$ *decrypts the ciphertext* $\mathbf{ct}$ *to a plaintext* $\mathbf{m}$

2.2.1 FHE Schemes

FHE schemes can be categorized in four consecutive generations. The first generation consists of the first fully homomorphic encryption introduced by Gentry [Gen09] (and improvements thereupon); while this scheme was a major theoretical breakthrough, it was wholly impractical.

B/FV, BGV. Second-generation schemes were derived from techniques by Brakerski, Gentry, Vaikuntanathan, and others. The two schemes of this generation still used extensively in practice today are B/FV [FV12] and BGV [BGV14].

In the simplest version of B/FV and BGV, the message space is $\mathbb{Z}_t$, and the ciphertext space is $(\mathbb{Z}_q[X]/\langle X^N+1\rangle)^k$ with $k \geq 2$. A more sophisticated version (which is actually used in practice) uses the *batching* technique to pack N messages in $\mathbb{Z}_t$ into one ciphertext, under the restriction that $t = 1 \mod 2N$. Effectively, this allows the message space to be extended to $\mathbb{Z}_t[X]/\langle X^N+1\rangle$. All FHE operations are then carried out "slot-wise", and one can view of the message space as an array of N values, with addition and multiplication being performed independently for each slot.

TFHE, FHEW. Third-generation schemes eliminate the expensive *relinearisation* procedure (see Section 2.2.2), and rely much more heavily on *bootstrapping*. TFHE [Chi+20] and FHEW [DM15] are the two major schemes from this generation that are used in practice. Here, the message space is $\mathbb{Z}_2$.

CKKS. In 2016, Cheon et al. introduced the CKKS scheme [Che+17], which supports approximate computations. CKKS is more efficient in practice than BGV and B/FV, even if it uses the same underlying rings for plaintexts and ciphertexts, and is the preferred scheme for machine learning applications.

2.2.2 FHE operations

FHE circuits are expressed as a directed acyclic graph of gates connected by wires. State-of-the-art FHE schemes support the following gates:

- Ciphertext-ciphertext addition (Add : $\mathcal{C} \times \mathcal{C} \to \mathcal{C}$) and ciphertext-plaintext (AddPlain : $\mathcal{C} \times \mathcal{M} \to \mathcal{C}$) addition, which does not require an evaluation key
- Ciphertext-ciphertext (Mul : $\mathcal{C} \times \mathcal{C} \to \mathcal{C}$) and ciphertext-plaintext (MulPlain : $\mathcal{C} \times \mathcal{M} \to \mathcal{C}$) multiplication, which does not require an evaluation key
- Modulus switching (ModSwitch : $\mathcal{C} \to \mathcal{C}$), which does not require an evaluation key

- Rotation by k ($\mathrm{Rot}_{\mathsf{ek}} : \mathcal{C} \times \mathbb{N} \to \mathcal{C}$), which requires a rotation key
- Relinearization ($\mathrm{Relin}_{\mathsf{ek}} : \mathcal{C} \to \mathcal{C}$), which requires a relinearization key
- Bootstrapping ($\mathrm{Bootstrap}_{\mathsf{ek}} : \mathcal{C} \to \mathcal{C}$), which requires a bootstrapping key

Operations involving a ciphertext and a plaintext are much faster than their ciphertext-ciphertext equivalent (due to the smaller representation of plaintexts), and are therefore used extensively in practice. In order for an FHE scheme to be useful, it should satisfy some notion of (possibly approximate) correctness; informally, performing operations on ciphertexts should translate to the same operations being performed on the underlying plaintexts. Additionally, some functionalities (like branching or looping on encrypted data) are not easily supported by FHE, which in practice limits FHE computations to logical or arithmetic circuits.

In order to illustrate how these operations are implemented in practice, we recall the definition of the BV scheme [BV11] below. The BV scheme serves as a blueprint for the modern B/FV, BGV, and CKKS schemes, and is often used in the literature to illustrate new constructions and attacks.

Definition 2.2 (BV scheme [BV11])
Let χ be a Gaussian distribution over the ring $R = \mathbb{Z}[X]/\langle X^N + 1\rangle$. For circuits of multiplicative depth up to degree d, the BV scheme is defined as follows:

$\mathsf{KGen}(1^\kappa)$: *returns a secret key* $\mathsf{sk} := (1, s, \ldots, s^{d-1})$*, where* $s \leftarrow \chi$*, and a public key* $\mathsf{pk} := (a, as + te)$*, where* $a \leftarrow R_q$ *and* $e \leftarrow \chi$*;*

$\mathsf{Enc}_(\mathbf{m})$: *samples* $u, e_0, e_1 \leftarrow \chi$*, and returns the ciphertext* $\mathbf{ct} = (\mathbf{ct}_0, \mathbf{ct}_1, 0, \ldots, 0)$ $\in R_q^d$*, where* $\mathbf{ct}_0 := \mathbf{m} + \mathsf{pk}_0 u + e_0$ *and* $\mathbf{ct}_1 := -\mathsf{pk}_1 u + e_1$*;*

$\mathsf{Eval}(f, \mathbf{ct}_1, \ldots, \mathbf{ct}_k)$: *proceeds gate-by-gate, computing additions and multiplications as follows:*

- $(\mathbf{ct} + \mathbf{ct}')_i := \mathbf{ct}_i + \mathbf{ct}'_i$
- $(\mathbf{ct} \cdot \mathbf{ct}')_i := \sum_{j=0}^{i} \mathbf{ct}_j \cdot \mathbf{ct}'_{j-i}$

$\mathsf{Dec}(\mathbf{ct})$: *computes* $[\langle \mathbf{ct}, \mathsf{sk}\rangle]_t = \left[\sum_{i=0}^{d-1} \mathbf{ct}_i \cdot s^i\right]_t$

The BV scheme does not support ciphertext maintenance operations (i.e., relinearisation, modulus switching, bootstrapping).

2.2.3 FHE Optimizations and Implementations

In order to make FHE efficient enough in practice, all implementations rely on a small set of optimizations, which allow for tremendous speed-ups at the cost of limitations on FHE parameters and operations.

Firstly, the ciphertext modulus q is usually several hundred bits long (for B/FV, BGV, and CKKS); in order to use standard CPU instructions, this modulus is split into a chain of prime moduli $q = \prod_{i=1}^{l} q_i$, where each q_i fits in a machine word (typically, $q_i \leq 2^{62}$). Each ciphertext coefficient is then split into l components using the Chinese Remainder Theorem (CRT), and is said to be in CRT or RNS (Residue Number System) representation.

One of the most elementary (and expensive) operations in FHE is the multiplication of two polynomials in R_q, which requires convoluting two polynomials of high degree $N \geq 2^{10}$, followed by a modular reduction by q and by $X^N + 1$. While doing this naïvely would have a complexity of $\Theta\left(N^2\right)$, implementations use the Fast Fourier Transform (FFT) (or rather the Number-Theoretic Transform (NTT), its equivalent for finite fields), which has a complexity of $\Theta\left(N \log(N)\right)$. This forces N to be a power of two (typically, $N \in \left\{2^k \,\middle|\, k \in [10..16]\right\}$).

Chapter 3

Decomposition and Analysis of FHE Integrity Constructions

Over the years, a number of approaches have been proposed to mitigate the integrity issues of FHE (some ad-hoc, and some more principled constructions). However, these approaches cover slightly different settings and provide different variants of integrity for a variety of settings. In this chapter, we aim to provide a generic framework for reasoning about integrity and privacy guarantees for FHE deployments, and to unify existing constructions into general paradigms. This allows us to fairly compare existing approaches, and to inform and simplify the design of newer and better integrity solutions.

In Section 3.1, we study existing integrity approaches for FHE and group them into four general paradigms. In Section 3.2, we present a unified formal definition of verifiable computation that generalises the various definitions used in the literature. Finally, we discuss limitations of existing instantiations and the inherent challenges posed by FHE integrity in Section 3.3.

3.1 Existing Paradigms for FHE Integrity

State-of-the-art FHE integrity approaches derive from a long branch of research that has tried to address the challenges of securely and privately offloading data (and computation thereon) to an untrusted server.

Gennaro et al. were the first to introduce the notion of Verifiable Computation (VC) [GGP10], which allows a client to outsource the computation of a function (on inputs provided by the client) to a server, with a guarantee that the result returned by the server would be correct. A VC scheme allows the client to encode its input x as (σ_x, τ_x), where σ_x is a public value to be sent to the server, and τ_x is kept by the client for later verification. The server can then evaluate the function on σ_x, yielding the encoded result σ_y, which is sent back to the client. Finally, the client can decide whether it accepts the result

σ_y (given the corresponding verification tag τ_x), and can then decode σ_y to y. Informally, this initial notion of VC requires the following guarantees:

Correctness: an honest client will always accept the result computed by an honest server;

Security: a malicious server cannot make a client accept a false result;

Input and Output Privacy: a malicious server does not learn anything about x or y from its interaction with the client;

Outsourceability: for the client, offloading should be (asymptotically) more efficient than computing the function (on unencoded inputs) in the first place.

While the connection between VC and FHE might not seem obvious at first, we will show in Section 3.2 that VC actually captures the vast majority of FHE integrity approaches in the literature. However, before introducing the general paradigms for FHE integrity, let us first delve a bit into the early VC literature (which often has few ties with FHE) in order to better understand how this notion came to be, and shed some light on some of its characteristics.

3.1.1 Early Verifiable Computation Approaches

This initial definition prompted a flurry of follow-up work, with a variety of VC constructions based on different cryptographic primitives.

In the same paper [GGP10], Gennaro et al. propose a VC scheme, by evaluating the outsourced function using Yao's garbled circuits [Yao82; Yao86]. Garbled circuits by themselves actually implement a one-time verifiable computation scheme, but the circuits need to be re-generated for each new computation request. To overcome this issue, Gennaro et al. use an FHE scheme in order to reuse the same garbled circuit multiple times (by encrypting the garbled circuit labels instead of revealing them to the server). Interestingly, FHE is not used here to protect the data directly, but rather to ensure that the garbled circuit remains secure under reuse.

A few years later, Parno et al. [PRV12] extended the VC definition to include *public delegatability* (meaning that every party is able to encode its input x as σ_x and τ_x), and *public verifiability* (every party is able to verify a result σ_y given a corresponding verification tag τ_x), and provide a construction that satisfies these new requirements. While this construction does not involve FHE (rather, it is based on attribute-based encryption), we mention this paper because the notions of public delegatability and verifiability it introduces have been prevalently used in later FHE-based VC constructions. In particular, [PRV12] is one of the few papers (if not the only paper) that present a concrete setting where public delegatability and verifiability are desirable.

Four parties are involved in this example scenario: a doctor (which we will call the organiser), a lab assistant (the encryptor), a server, and a patient (the verifier). Public delegatability would allow the organiser to bear the burden of an expensive, one-time pre-processing phase (and to determine the specifics of the function to be offloaded). After this setup phase, any encryptor could use the function-specific public and evaluation keys to delegate computations to the server, with no involvement from the organiser. Public verifiability, on the other hand, allows the encryptor to also generate a verification key when they delegate the computation to the server. The verifier can then obtain the output (i.e., their test result) directly from the server, and verify it using the verification key it received from an encryptor.

However, the construction in [PRV12] does not achieve input privacy[1]. When considering VC schemes with input and output privacy, it is unclear in which settings public verifiability would be sensible. On one hand, the ability to verify a result that one cannot decrypt seems quite artificial (e.g., the patient in the setting above would only learn that there exists a valid encoding of their test result, but nothing more). On the other hand, a party that needs to decode the result would already need to possess of a secret key, and it is unclear how to meaningfully define a key generation protocol in this setting. One remaining benefit of public verifiability is that it naturally grants protection against verification oracle (as the VC scheme securely supports verification by any party, even the adversary). However, protection against verification queries can also be achieved through other means (e.g., by aborting on invalid verifications). We investigate this discrepancy further in Section 5.1.3 and show that for most modern FHE use cases, public verifiability is not required.

Some additional non-FHE approaches to construct (input-private) VC were also proposed, e.g., by Goldwasser et al. in [Gol+13] (based on a single-key functional encryption). However, these approaches are very limited (e.g., [Gol+13] does not guarantee output privacy, and only supports single-bit outputs). Since then, FHE-based approaches (which are often more flexible and offer more guarantees) have formed the bulk of the research on securely delegating computation. In the remainder of this section, we will study the existing literature on FHE integrity, and classify approaches into four paradigms: MAC-then-Encrypt, Encrypt-and-MAC, Encrypt-then-MAC, and Compute-then-Prove.

3.1.2 MAC-then-Encrypt

The MAC-then-Encrypt (MtE) paradigm requires users to first compute a (homomorphic) MAC over their plaintexts before encrypting both with FHE. If special care is taken to ensure that the FHE homomorphism applies both

[1]Parno et al. remark that by using attribute-hiding attribute-based encryption, their construction could achieve input privacy [PRV12, Remark 1], but the authors do not define their notion of input privacy. In particular, the relation between input privacy and public verifiability is not investigated.

to the plaintext and its MAC, a result can be decrypted to a message and its MAC, which can then be checked. Here, the privacy of the MtE scheme can be reduced to the semantic security of the FHE scheme, and security to the unforgeability of the MAC. However, special care must be taken in cases where the user might implement an oracle (e.g., a MAC verification oracle), which could leak information about the plaintext. This is the reason some MtE approaches are not secure against verification queries. As for more traditional public-key encryption schemes, this is usually addressed by relying on the MAC secret, and/or by hiding the positions of the MAC and plaintext in the ciphertext using FHE's semantic security.

The MtE paradigm for FHE integrity was first used by Gennaro and Wichs [GW13]; for a security parameter κ, each bit b of plaintext is encrypted to κ ciphertexts $\mathbf{ct}_1, \ldots, \mathbf{ct}_\kappa$, with $\mathbf{ct}_i$ is either an encryption of b (if i is a randomly selected half of the set $\{1, \ldots, \kappa\}$, or a pseudo-random encryption of 0 with random coins derived from a PRF (if i is in the other half). This construction is not secure against verification queries. It is also not efficiently verifiable, as re-computing the result MAC after the computation is as expensive as computing the result in the first place. To remedy this, Gennaro and Wichs propose to use a Succinct Non-interactive ARGument (SNARG) (for relations in P) to allow for a more efficient verification procedure, but give few additional details. This construction is not secure against verification queries, but can be made secure for an a-priori bounded number of queries [GW13].

Following up on this work, Catalano and Fiore introduce a new information-theoretic MAC [CF13], which supports fewer functions (only arithmetic circuits of a bounded depth) but is more efficient. For each plaintext $\mathbf{m}$, the MAC is a degree-1 polynomial that equals $\mathbf{m}$ when evaluated at 0, and equals a pseudo-random value when evaluated at a random point. However, they only present their MAC construction, and do not study its composition with FHE.

More recently, Chatel et al. [Cha+22] have generalised these two approaches and adapted them for state-of-the-art FHE schemes, and provided the first implementation of such a scheme. As the MAC used in the MtE paradigm only needs to support homomorphism with respect to plaintexts, it is relatively easy to support all (ciphertext) operations of state-of-the-art FHE schemes; [Cha+22], for example, support rotations and relinearisations. Similarly to [GW13], this approach is not secure against verification queries.

While the notion of these so-called *homomorphic authenticator* approaches is not quite equivalent to VC, we explore the connection between the two notions in Section 3.2.

3.1.3 Encrypt-and-MAC

In the Encrypt-and-MAC (EaM) paradigm, a MAC is computed on the plaintext, and the plaintext is encrypted using the original FHE scheme. In order to ensure privacy, the MAC must be semantically secure as well. Additionally, some care must be taken when evaluating, as both MACs and ciphertexts must undergo the same operations, although they may be elements of different arithmetic structures. In particular, ciphertext maintenance operations (which modify the ciphertext without modifying the underlying plaintext) must have an equivalent for MACs.

In [LWZ18], Li et al. introduce the notion of privacy-preserving homomorphic MACs, which satisfy this hiding property, by using leveled multilinear maps, and a generic FHE scheme.

3.1.4 Encrypt-then-MAC

The Encrypt-then-MAC (EtM) approach firsts encrypts the plaintext, which is then protects with a homomorphic MAC before outsourcing. As the MAC does not depend on the plaintext, the privacy of the scheme can be reduced to the semantic security of the underlying FHE scheme, while its security hinges on the unforgeability of the MAC. While conceptually easier to analyse than the MtE and EaM, the MAC in EaM constructions needs to be homomorphic *with respect to operations on ciphertexts* (including ciphertext maintenance operations), which is often much harder to achieve than the corresponding homomorphic property on plaintexts.

This limits the applicability of this approach in practice; Fiore et al. make use of this paradigm in [FGP14], and instantiate their MAC using pairings. In order to bridge the gap between ciphertexts in FHE rings and the pairing groups, they introduce a homomorphic hash function from R_q to $\mathbb{Z}_q$ (where q prime is the order of the pairing group). The efficiency of the verification is guaranteed by using *amortised closed-form efficient PRFs*. However, the combination of these primitives to construct the homomorphic MAC limits the applicability of the [FGP14] approach to quadratic circuits only.

3.1.5 Compute-then-Prove

A much newer paradigm, which we dub *Compute-then-Prove* (CtP) replaces MACs with (zero-knowledge) proofs (or rather arguments). Here, everything is the same as for a traditional FHE application, except that the server must additionally provide a proof that the output ciphertext is the result of applying some (usually public) circuit to the client's inputs. This paradigm addresses issues from the other three paradigms presented above: Firstly, proving over ciphertexts allows for easier security proofs, and typically stronger security guarantees against verification oracles. Additionally, proofs are more flexible than MACs, and allow for *non-deterministic* functions, where the server is allowed to contribute to the computation with a (private) input. At least in theory, modern proof systems (e.g., Succinct Non-interactive ARguments of Knowledge (SNARKs)) should also offer a competitive efficiency compared to MACs, but the concrete efficiency of such proof systems has not fully caught up yet.

Fiore et al. first introduced a SNARK for FHE in [FNP20], with support for arbitrarily deep computations on the BV scheme. However, the SNARK and FHE parameters of this construction need to be compatible, which puts unrealistic assumptions on the FHE parameters (e.g., the ciphertext modulus q should be a big prime). Using their newly introduced SNARK, Fiore achieve *public delegatability* and *public verifiability.* Additionally, they introduce the notion of *context-hiding*, which preserves input privacy against a party that has access to the result ciphertext and any verification information about the input.

Following up on this work, Bois et al. introduced a more flexible instantiation of the CtP paradigm, by decoupling the SNARK and FHE components further (by using homomorphic hashing to map FHE ciphertexts to inputs more suitable for the SNARK). This construction achieves public delegatability and public verifiability. This is also the first (and to the best of our knowledge the only) VC approach capable of handling non-deterministic computation, and to offer provable security for this setting. However, their construction is only applicable to the BV scheme (although with general modulus choices). Even more recently, Ganesh et al. construct a new SNARK over rings, which can more easily represent FHE computations; however, it is not specified how one would create a VC scheme from this SNARK.

Finally, Natarajan et al. [Nat+21] explored the use of hardware Trusted Execution Environments (TEEs) as a proving primitive. Here, a hardware attestation would take on the role of a cryptographic proof or argument, while input privacy is guaranteed by FHE. Natarajan et al. show how this approach can support non-deterministic inputs, and implement their solution for state-of-the-art FHE schemes.

While they are at the moment still more limited (both in terms of efficiency and expressivity) than other constructions (e.g., [Cha+22]), approaches using the Compute-then-Prove paradigm are the only ones that can support real-world non-deterministic FHE use cases with inputs from multiple parties. As we believe these approaches are the most promising to achieve flexible integrity guarantees for FHE, we will therefore focus on this paradigm for the remainder of this thesis.

3.2 Integrity through Verifiable Computation

We now present a unified formal definition of verifiable computation for FHE, which captures and generalises the variants we observed in the literature. As we gain a more consolidated view of the field of VC for FHE, we point out gaps and definitional limitations common throughout the literature, which we will discuss in Chapter 5.

Definition 3.1 (Verifiable Computation)
A verifiable computation *scheme* $\mathcal{VC} = (\mathsf{KGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ *consists of the following PPT Turing machines:*

$\mathsf{Setup}(1^\kappa) \to (\mathsf{pk}, \mathsf{sk})$: *for a security parameter κ, generates the function-independent public and private parameters;*

$\mathsf{KGen}_{\mathsf{pk}}(f) \to (\mathsf{pk}_f, \mathsf{sk}_f)$: *generates a public key (used by the server to compute f) and a secret key (used for verifying and decoding the result);*

$\mathsf{ProbGen}_{\mathsf{key}}(x) \to (\sigma_x, \tau_x)$: *using* key*, encode the input x as a public value σ_x (given to the server), and a verification tag τ_x. If* key *is* pk*, the VC scheme is said to offer* public delegatability *[PRV12], i.e., anyone can delegate computations. On the other hand, if* key *is* sk*, we will say that the VC scheme offers* designated delegatability*;*

$\mathsf{Compute}_{\mathsf{pk}_f}(\sigma_x, w) \to \sigma_y$: *computes an encoded version of the function's output $y = f(x, w)$, given the server's input w;*

$\mathsf{Verify}_{\mathsf{key}}(\tau_x, \sigma_y) \to b$: *verify the result of the computation, and decode; the client accepts $y = f(x, w)$ if $b = 1$, and rejects otherwise. If* key *is* pk_f*, the VC scheme is said to offer* public verifiability*, and offers* designated verifiability *if* key *is* sk_f*;*

$\mathsf{Decode}_{\mathsf{sk},\mathsf{sk}_f}(\sigma_y) \to y$: *decodes σ_y to y using the secret keys.*

If the server can contribute to the computation with an input w, we say that the VC scheme supports non-determinism*; if $w = \emptyset$, the scheme is deterministic.*

A VC scheme must satisfy the correctness, outsourceability, security*, and* input privacy *properties, as defined below.*

The notion of non-deterministic VC was introduced by Bois et al. [Boi+21] (who however only formulate it in conjunction with public delegatability and public verifiability), and naturally generalises the non-deterministic case. In the following, we will use key as a generic placeholder for either a public or secret key, depending on whether a given scheme offers public delegatability/verifiability.

Definition 3.2 (Correctness)
Correctness guarantees that an honest verifier will always accept a result computed by an honest worker. More formally, for all functions f, and for all (x, w) in the domain of f:

$$\Pr\left[\begin{array}{c|c} \mathsf{Verify}_{\mathsf{key}}(\tau_x, \sigma_y) = 1 & (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\kappa) \\ \wedge & (\mathsf{pk}_f, \mathsf{sk}_f) \leftarrow \mathsf{KGen}(f) \\ & (\sigma_x, \tau_x) \leftarrow \mathsf{ProbGen}_{\mathsf{key}}(x) \\ \mathsf{Decode}_{\mathsf{sk},\mathsf{sk}_f}(\sigma_y) = f(x, w) & \sigma_y \leftarrow \mathsf{Compute}_{\mathsf{pk}_f}(\sigma_x, w) \end{array}\right] = 1$$

We note here that this intuitive definition assumes that the VC scheme is both *exact* (the decoded y must match $f(x, w)$ exactly), and *perfect* (the decoding step must always succeed, and is not allowed to fail with negligible probability). These assumptions de facto prevent instantiations with some modern FHE schemes, as we show in Section 5.3.

Definition 3.3 (Outsourceability)
Outsourceability ensures that the complexity of the computations performed by the verifier (namely $\mathsf{ProbGen}$ *and* Verify*) is less than the computation required to evaluate f (over the unencoded input x). Formally, for any x and σ_y, the time required by $(\sigma_x, \tau_x) \leftarrow \mathsf{ProbGen}_{\mathsf{key}}(x)$ plus the time required by $\mathsf{Verify}_{\mathsf{key}}(\tau_x, \sigma_x)$ is in $o(T)$, where T is the time required to compute $f(x)$.*

Contrary to [Boi+21], we only define outsourceability for deterministic schemes; indeed, the requirement that computations performed by the client take less time than computing $f(x, w)$ is ill-defined when the client does not have access to w. We therefore waive the outsourceability requirement for the non-deterministic case.

Definition 3.4 (Security)
A VC scheme is secure if a malicious worker cannot make the verifier accept an incorrect answer. Formally, a VC scheme is secure if for any PPT adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following game $\mathsf{Expr}^{\mathsf{Ver}}[\mathcal{A}](1^\kappa)$ is negligible:

$$\mathsf{Adv}^{\mathsf{Ver}}[\mathcal{A}](\kappa) = \Pr\Big[\mathsf{Expr}^{\mathsf{Ver}}[\mathcal{A}](1^\kappa) \Rightarrow 1\Big] = \mathsf{negl}(\kappa)$$

```
$\mathsf{Expr}^{\mathsf{Ver}}[\mathcal{A}](1^\kappa)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\kappa)$
$x \leftarrow \mathcal{A}_1^{\mathcal{O}_1}(\mathsf{pk})$
$(\sigma_x, \tau_x) \leftarrow \mathsf{ProbGen}_{\mathsf{key}}(x)$
$(f, \sigma_y) \leftarrow \mathcal{A}_2^{\mathcal{O}_2}(\sigma_x, \tau_x)$
$b \leftarrow \mathsf{Verify}_{\mathsf{key}}(\tau_x, \sigma_y)$
$\textbf{return } (b = 1) \wedge \Big(\nexists w : \mathsf{Decode}_{\mathsf{sk},\mathsf{sk}_f}(\sigma_y) = f(x, w)\Big)$
```

The set of oracles $\mathcal{O}_1, \mathcal{O}_2$ available to the adversary (chosen from the oracles below) determines the strength of the adversarial model. Usually, the minimum set of oracle considered in the literature is $\mathcal{O}_1 = \mathcal{O}_2 = \{\mathcal{O}_{\mathsf{KGen}}, \mathcal{O}_{\mathsf{ProbGen}}\}$, where $\mathcal{O}_{\mathsf{KGen}}$ is only allowed to be queried once[2].

VC schemes with security against verification queries additionally grant the adversary access to the verification oracle $\mathcal{O}_{\mathsf{Verify}}$.

```
$\mathcal{O}^{\mathsf{Ver}}_{\mathsf{KGen}}(f)$   // Only queried once
$(\mathsf{pk}_f, \mathsf{sk}_f) \leftarrow \mathsf{KGen}_{\mathsf{key}}(f)$
return $\mathsf{pk}_f$
```

```
$\mathcal{O}^{\mathsf{Ver}}_{\mathsf{ProbGen}}(x)$
$(\sigma_x, \tau_x) \leftarrow \mathsf{ProbGen}_{\mathsf{key}}(x)$
return $\sigma_x$
```

```
$\mathcal{O}^{\mathsf{Ver}}_{\mathsf{Verify}}(\tau, \sigma)$
$b \leftarrow \mathsf{Verify}_{\mathsf{key}}(\tau, \sigma)$
return $b$
```

```
$\mathcal{O}^{\mathsf{Ver}}_{\mathsf{Decode}}(\tau, \sigma)$
$b \leftarrow \mathsf{Verify}_{\mathsf{key}}(\tau, \sigma)$
if $b = 0$ :
  return $\perp$
else :
  return $\mathsf{Decode}_{\mathsf{sk},\mathsf{sk}_f}(\sigma)$
```

Definition 3.4 defines security for the non-deterministic case (as in [Boi+21]); for the deterministic case, $\mathcal{A}$'s winning condition simplifies to $(b = 1) \wedge (\mathsf{Decode}_{\mathsf{sk},\mathsf{sk}_f}(\sigma_y) \neq f(x))$. We note here that this definition does not allow for approximate VC schemes, which would be trivially insecure under this definition (the decoded value y would only be approximately $f(x)$, but not exactly). This is a similar issue as in Definition 3.2, and we resolve it in Section 5.3.

The oracle $\mathcal{O}_{\mathsf{ProbGen}}$ is needed to model non-publicly-delegatable schemes (and is redundant for publicly-delegatable schemes), as the adversary would otherwise not be able to generate encodings [FNP20]. For schemes with designated delegatability, the secret verification value τ_x is not shared with the adversary.

We also introduce the decoding oracle $\mathcal{O}_{\mathsf{Decode}}$, which naturally generalises the verification oracle $\mathcal{O}_{\mathsf{Verify}}$. Indeed, it seems quite artificial to us to restrict the adversary to the verification stage only. To the best of our knowledge, none of the VC schemes for FHE outlined above consider such a decoding oracle. We show how this omission can to complete key-recovery attacks and a total loss of privacy in Chapter 5.

[2]This is done for simplicity; indeed, any adversary calling $\mathcal{O}_{\mathsf{KGen}}$ multiple times can be reduced straightforwardly to an adversary making only one such query, and can be easily simulated [Boi+21].

Definition 3.5 (Input Privacy)
Input privacy is formalised using an indistinguishability game: formally, a VC scheme provides input privacy if the advantage of any PPT adversary $\mathcal{A}$ in the following game $\mathsf{Expr}^{\mathsf{Priv}}[\mathcal{A}](1^\kappa)$ is negligible:

$$\mathsf{Adv}^{\mathsf{Priv}}[\mathcal{A}](\kappa) = \left|\Pr\left[\mathsf{Expr}_0^{\mathsf{Priv}}[\mathcal{A}](1^\kappa) \Rightarrow 0\right] - \Pr\left[\mathsf{Expr}_1^{\mathsf{Priv}}[\mathcal{A}](1^\kappa) \Rightarrow 1\right]\right| = \mathsf{negl}(\kappa)$$

```
$\mathsf{Expr}_b^{\mathsf{Priv}}[\mathcal{A}](1^\kappa)$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\kappa)$
$(x_0, x_1) \leftarrow \mathcal{A}_1^{\mathcal{O}_1}(\mathsf{pk}, 1^\kappa)$
$(\sigma_b, \tau_b) \leftarrow \mathsf{ProbGen}_{\mathsf{key}}(x_b)$
$\widehat{b} \leftarrow \mathcal{A}_2^{\mathcal{O}_2}(\mathsf{pk}, \sigma_b, \tau_b)$   // Respectively $\mathcal{A}_2^{\mathcal{O}_2}(\mathsf{pk}, \sigma_b)$
return $b = \widehat{b}$
```

The set of allowed oracles $\mathcal{O}_1, \mathcal{O}_2$ in this game are the same as in Definition 3.4, and the oracles themselves are implemented in the same way, except for $\mathcal{O}_{\mathsf{Decode}}$, which would need to be adapted to prevent trivial attacks (i.e., the adversary should not be allowed to query the decoding of σ_b, or any σ derived from σ_b). However, as $\mathcal{O}_{\mathsf{Decode}}$ is usually not part of $\mathcal{O}_2$ (due to the impossibility of achieving CCA2 security, see Chapter 6), we do not write out these checks.

We define two different versions of the experiment, varying the inputs given to $\mathcal{A}_2$ in the post-challenge phase: in approaches with public verifiability, the adversary receives $(\mathsf{pk}, \sigma_b, \tau_b)$ (capturing the requirement that a verification tag τ_x should not leak information about x), whereas schemes with designated verifiability only give access to σ_b.

Finally, we specify the context-hiding property, as introduced in [FNP20] and extended to the non-deterministic case in [Boi+21].

Definition 3.6 (Context-Hiding)
Informally context-hiding guarantees two properties:

(a) An adversary with access to $\tau_x, \mathsf{sk}, \mathsf{sk}_f$, and σ_y should not learn anything more about (x, w) than what it could not infer by being honest. This models the case where the verifier/decryptor is a different party than the encryptor (e.g., an analyst that is only authorised to decrypt aggregated analytics of private data, but not the raw data).

(b) An adversary with access to $\tau_x, \mathsf{sk}, \mathsf{sk}_f, \sigma_y$ and σ_x should not learn anything more about w than what it could infer by being honest. This models the case where the verifier/decryptor colludes with the encryptor (or is the same party), and tries to recover the server's private input.

Formally, context-hiding is defined using simulators: a $\mathcal{VC}$ scheme is context-hiding if there exist efficient simulator algorithms S_1, S_2, S_3, S_τ such that:

S_1 : *the following distributions are indistinguishable:*

$$\{(\mathsf{pk}, \mathsf{sk}) \mid (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\kappa)\}$$

$$\{(\mathsf{pk}^*, \mathsf{sk}^*) \mid (\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{td}) \leftarrow S_1(1^\kappa)\}$$

S_2 : *for any f, the following distributions are indistinguishable:*

$$\left\{(\mathsf{pk}_f, \mathsf{sk}_f) \;\middle|\; \begin{array}{c}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\kappa)\\ (\mathsf{pk}_f, \mathsf{sk}_f) \leftarrow \mathsf{KGen}_{\mathsf{pk}}(f)\end{array}\right\}$$

$$\left\{(\mathsf{pk}^*_f, \mathsf{sk}^*_f) \;\middle|\; \begin{array}{c}(\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{td}) \leftarrow S_1(1^\kappa)\\ (\mathsf{pk}^*_f, \mathsf{sk}^*_f, \mathsf{td}_f) \leftarrow S_2(\mathsf{td}, f)\end{array}\right\}$$

S_3 : *For any f and (x, w), the following distributions are indistinguishable:*

$$\left\{(\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{pk}^*_f, \mathsf{sk}^*_f, \sigma_x, \tau_x, \sigma_y) \;\middle|\; \begin{array}{c}(\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{td}) \leftarrow S_1(1^\kappa)\\ (\mathsf{pk}^*_f, \mathsf{sk}^*_f, \mathsf{td}_f) \leftarrow S_2(\mathsf{td}, f)\\ (\sigma_x, \tau_x) \leftarrow \mathsf{ProbGen}_{\mathsf{key}}(x)\\ \sigma_y \leftarrow \mathsf{Compute}_{\mathsf{pk}_f}(\sigma_x, w)\end{array}\right\}$$

$$\left\{(\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{pk}^*_f, \mathsf{sk}^*_f, \sigma_x, \tau_x, \sigma^*_y) \;\middle|\; \begin{array}{c}(\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{td}) \leftarrow S_1(1^\kappa)\\ (\mathsf{pk}^*_f, \mathsf{sk}^*_f, \mathsf{td}_f) \leftarrow S_2(\mathsf{td}, f)\\ \sigma^*_y \leftarrow S_3(\mathsf{td}_f, \tau_x, f(x, w))\end{array}\right\}$$

S_τ : *for any x and for $(\sigma_x, \tau_x) \leftarrow \mathsf{ProbGen}_{\mathsf{key}}(x)$, $S_\tau(\mathsf{td})$ is indistinguishable from τ_x.*

3.2.1 Verifiable Computation: a Unified FHE Integrity Notion

Some approaches studied in Section 3.1 have been introduced using some variant of the verifiable computation formalism, and are thus captured directly by our definitions above (this is the case for all cryptographic constructions using the EtM and CtP paradigms). Other works (in particular those using the MtE and EaM paradigms) have coined their own terminology and definitions. In this subsection, we connect these constructions to verifiable computation, and express them using the formal definitions above.

MAC-then-Encrypt. We formalise MAC-then-Encrypt schemes as deterministic, privately-delegatable and privately-verifiable VC schemes. Here, KGen is independent of the function, and generates a private MAC key, and an empty public key. For each plaintext in the input x, ProbGen firsts computes a MAC on both the plaintext and on its identifier (e.g., the index of this plaintext among the inputs to the computation), and encrypts both the plaintext and the MAC as σ_x. The verification tag τ_x is simply the index of the plaintext, and Compute is simply the FHE evaluation function. For verification, σ_y is first decrypted to a resulting plaintext and its corresponding MAC; this MAC-plaintext pair can then be checked by the client using its secret MAC key. Finally, Decode simply outputs the plaintext, stripped from its MAC. Since all inputs to the computation need to be MAC-ed first in order for the verification to go through, these schemes do not support inputs from the server.

The fact that for σ_y needs to first be decrypted before any verification can take place, coupled with the MAC being computed on private information, makes this approach prone to leak secret information upon verification, and is therefore usually not secure against verification oracles.

Encrypt-and-MAC. We formalise Encrypt-and-MAC schemes as deterministic, privately-delegatable and publicly-delegatable VC schemes. As for MtE schemes, KGen generates MAC keys independently of the function to be evaluated. For an input x, ProbGen MACs all plaintexts and their indices, and encrypts them separately. The encoding σ_x is thus composed of pairs of MACs and ciphertexts, while the verification tag τ_x only holds the plaintext indices. Compute performs the computation on both ciphertexts and MACs simultaneously, and outputs a ciphertext together with its corresponding MAC. For verification, the resulting ciphertext is first decrypted, and the plaintext is checked against the MAC. Decoding simply returns the plaintext if the verification succeeded.

As for the MtE case, this approach is prone to leak secret information upon verification, as the resulting ciphertext must also be decrypted before any checks take place, and the MAC is also dependent on the ciphertext. For example, we note that while the approach in [LWZ18] is secure (in the sense of Definition 3.4) against verification queries, its input privacy is not guaranteed in the presence of a verification oracle.

3.3 Limitations of Existing Paradigms and Instantiations

Unfortunately, throughout all these FHE integrity approaches (summarised in Table 3.1), we still see significant gaps between theoretical constructions and real-world FHE.

Approach		SotA FHE	Circuit	Non-Det.	Context-Hiding	Approx.	Verif. Queries	Impl.
MtE	[GW13]		Any	○	○	○	Bounded	○
	[CF13]		Any	○	○	○	Unbounded	○
	[Cha+22]	●	Any	○	○	○	None	●
EtM	[FGP14]	◑	Quadratic	○	○	○	None	●
EaM	[LWZ18]		Any	○	○	○	None	○
CtP	[FNP20]	◑	Any	○	●	○	Unbounded	○
	[Boi+21]	◑	LogspaceUnif	●	●	○	Unbounded	○
	[GNSV21]	◑	Any	●	●	○	None	◑†
	[Nat+21]	●	Any	●	○	●	Unbounded	●‡

†: implemented as part of this thesis, see Section 4.1.
‡: re-implemented as an open-source library as part of this thesis, see Section 4.2.

SotA FHE: level of support for state-of-the-art FHE schemes (◑ denotes support for the BV scheme (potentially with additional limitations), ● denotes full support for at least one scheme in {B/FV, BGV, TFHE, FHEW, CKKS}, including ciphertext maintenance operations, and an empty value denotes that the approach is generic, and was not evaluated with respect to a particular scheme);

Circuit: supported circuits ([Boi+21] only supports logspace-uniform circuit);

Non-Det.: support for a server input during the computation;

Context-Hiding: Support for context-hiding;

Approx.: support for approximate computations;

Verif. Queries: number of verification queries for which the scheme remains secure and input-private;

Impl.: whether the scheme was implemented and evaluated in practice (for [GNSV21], ◑ denotes that the implementation was carried out as part of this thesis, but was not part of the original paper).

Table 3.1: Characteristics and limitations of existing FHE integrity paradigms and approaches

First, most integrity approaches only offer support for a limited set of use cases. For example, paradigms involving MACs (MtE, EaM, EtM) are limited to authenticated analytics, and are therefore not suitable for more complex FHE deployments that involve inputs from another party. Similarly, context-hiding is achieved generically by only three approaches [FNP20; Boi+21; GNSV21]. Existing approaches are also limited in their support for general circuits, and in their support for approximate computation.

Additionally, even the most promising constructions based on the CtP paradigm are still limited in their support for state-of-the-art FHE schemes, as well as in their efficiency. In particular, none of the cryptographic CtP approaches support modern FHE schemes as implemented and used in practice.

On a more abstract note, many of these papers do not introduce a generic construction, but propose new primitives and a corresponding VC scheme at the same time, which leads to ad-hoc proofs and analyses tailored to a specific primitive. This makes generalising existing results and improving individual primitives harder than necessary. The subtle differences in VC definitions and adversarial models also hinder a fair comparison of the security guarantees of each approach. Besides, many works do not provide an (open-source) implementation of their approach, which significantly increases the difficulty of comparing existing approaches, let alone using integrity protection in practice.

Finally, in addition to this gap in functionality and efficiency between existing approaches and real-world FHE, we also see a significant gap in the adversarial models used in the VC community, and realistic FHE deployments. Most notably, verification oracles only crudely model the plethora of leakage avenues of a realistic application. We will investigate this issue in more detail in Chapter 5, and show concrete key-recovery attacks for such settings.

Beyond the inherent limitations of the paradigm being used, FHE approaches are also limited (both in efficiency and expressiveness) by the integrity primitives they use. In the next chapter, we outline the requirements needed for practical integrity primitives, and provide analyses, improvements, and an implementation for two concrete primitives.

Chapter 4

Evaluating Integrity Primitives in Practice

We will now take a closer look at the *integrity primitives* at the heart of the constructions outlined in Chapter 3, i.e., hardware-based or cryptographic mechanisms that ensure the integrity of a computation. In order to be practical, an integrity primitive (e.g., a zero-knowledge proof system) must overcome the following challenges:

Ciphertext Expansion. FHE ciphertexts are typically much larger than plaintexts (usually 1–2 orders of magnitude), which poses inefficiency issues when trying to prove properties over plaintexts by working over ciphertexts. In addition, some applications require strong efficiency guarantees, where the cost of offloading and verifying a computation to a server must be faster than performing this computation over plaintexts.

Algebraic Structure. State-of-the-art FHE schemes use specific algebraic structures (power-of-two cyclotomic rings with composite moduli), whereas most integrity and proving tools in the literature work over prime fields. In addition, efficient FHE implementations can use multiple rings for a single FHE operation (for example, multiplication in BGV uses three rings with different moduli), whereas non-FHE approaches to integrity usually only use the same structure throughout. While it is possible to bridge the gap between FHE structures and structures more amenable for proving, this generally incurs a heavy cost in efficiency or expressivity, or both.

Non-algebraic Operations. Efficient FHE implementations also make use of non-algebraic operations (e.g., rounding, bit-decomposition). These operations are hard to express natively in the language of existing proof systems, which again drastically limits the efficiency and/or expressivity of integrity approaches for state-of-the-art FHE schemes.

As discussed in Section 3.1, most integrity primitives used in the literature are either not expressive enough to capture the full capabilities of modern FHE, or not efficient enough to be realistically used in practice. These shortcomings prompt us to search for more practical integrity primitives for state-of-the-art FHE. In order to address the challenges outlined above, we conjecture that efficient and flexible integrity primitives for FHE should satisfy the following properties:

FHE-Friendliness. Each integrity primitive uses its own representation to express computations and statements; ensuring that the "translation gap" from FHE computations to the primitive's representation is minimal (e.g., by ensuring that they both operate on the same arithmetic structures, and use similar arithmetic operations) is a key factor for the concrete efficiency of the overall VC scheme. We note that this can also be achieved by tweaking the FHE scheme (e.g., by removing rounding operations), but this usually requires forfeiting existing optimizations for FHE.

Expressiveness through Flexibility. In order to cope with different FHE schemes (and different optimizations of the same scheme), the primitive should be general enough to accommodate these slight differences; e.g., supporting rings with different types of moduli, or supporting several rings in the same circuit. Note that this might conflict with the FHE-Friendliness requirement.

Efficiency through Shared Underlying Computations. To allow the integrity primitive to benefit from any speed-ups in the implementation of the ring arithmetic underlying the FHE library, it should be easy to express the primitive's computation as fundamental ring operations, which are also the lowest abstraction used by the FHE library (e.g., NTT-optimised multiplication). Simply forcing the primitive to operate on the same arithmetic structure as the FHE computation would not necessarily achieve this goal; rather, one should take care to make a concrete implementation of the primitive take advantage of the optimised FHE implementation.

Having defined these desirable criteria, we now introduce two families of integrity primitives that satisfy these criteria, and are thus promising candidates for efficient and flexible FHE VC schemes: ring-based zkSNARKs (Section 4.1), and hardware-backed Trusted Execution Environment (TEE) (Section 4.2). For both of these primitive families, we analyse their concrete complexity and expressiveness, introduce new theoretical efficiency improvements, and provide the first open-source (prototype) implementation, together with an empirical evaluation.

4.1 Argument Systems over Rings

While there have been many SNARKs over fields, Rinocchio [GNSV21] is the first SNARK working natively over rings. In particular, in [GNSV21] Ganesh et al. detail how Rinocchio can be used to prove FHE computations, which is why we believe it is one of the strongest candidate primitive for practical FHE integrity.

4.1.1 Analysing Complexity & Expressiveness

Rinocchio produces constant-sized proofs (consisting of 9 elements), and relies on linear-only encodings for security. Rinocchio requires the arithmetic circuit under consideration to be expressed as a Quadratic Ring Program (QRP). Let F be an arithmetic circuit with multiplicative depth d over the ring R with public input indices $I_{\text{io}} = 1, \ldots, i$ (corresponding to the client's public ciphertexts) and intermediate / private inputs $I_{\text{mid}} = i+1, \ldots, m$ (corresponding to the server's inputs and intermediate values). F is encoded as a QRP as follows: first, we sample a random element $r_i \in A$ for each gate. For each k-th value in $I_{\text{io}} \cup I_{\text{mid}}$, we define a polynomial $v_k(x)$ by interpolating, such that $v_k(r_i) = 1$ if the k-th value is a left input to the i-th multiplication gate, and 0 otherwise. This process is then repeated for w_k (encoding right inputs) and y_k (encoding outputs). Finally, $t(x)$ is set to $\prod_i (x - r_i)$.

The QRP is then the tuple $(t(x), (v_k(x), w_k(x), y_k(x))_{k=1}^m)$, where all elements of the tuple are univariate polynomials over R, with coefficients in A[1]. For all k, v_k, w_k, and y_k have degree at most $d-1$, and t has degree d, where d is the number of multiplications in the circuit.

The protocol relies on (probabilistically) checking that $t(x)$ divides $v(x) \cdot w(x) - y(x)$, which holds if and only if the values on the wires correspond to a satisfying circuit. Here, $v(x) = \sum_{k \in I_{\text{io}} \cup I_{\text{mid}}} a_k \cdot v_k(x)$, where $a_k \in R$ is the k-th circuit value. During the protocol, the quotient polynomial $h(x) = \frac{v(x) \cdot w(x) - y(x)}{t(x)}$ is used; h has degree at most $d-1$, and has coefficients in R. In the following, $A^* = A \setminus \{r_i\}_{i=1}^d$ is the set of values in the exceptional set that were not used to define the QRP, and R^* are the units of R.

We reproduce the Rinocchio protocol in Figure 4.1. In [GNSV21], the protocol is described using a very terse and abstract syntax, while we use a more precise notation that more closely matches how one would implement the protocol in practice; this allows us to make the concrete complexity of the protocol more explicit. We refer the interested reader to the full paper [GNSV21] for details.

[1]For simplicity, we drop the 0-th terms v_0, w_0, and y_0, which by convention correspond to the public constant input 1 (the unit in R).

```
Setup $(1^\kappa)$

$s \leftarrow A^*;\ \alpha \leftarrow R^*;\ r_v, r_w \leftarrow R^*;\ r_y = r_v \cdot r_w;\ \beta \leftarrow R \setminus \{0\}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{E.KGen}(1^\kappa)$
$\mathsf{crs} := \left(\left\{\mathsf{E}\left(s^i\right)\right\}_{i=1}^d, \left\{\mathsf{E}\left(\alpha s^i\right)\right\}_{i=1}^d, \left\{\mathsf{E}\left(\beta(r_v v_k(s) + r_w w_k(s) + r_y y_k(s))\right)\right\}_{k \in I_{\text{mid}}}, \mathsf{pk}\right)$
$\mathsf{vk} := (\mathsf{sk}, \mathsf{crs}, s, \alpha, \beta, r_v, r_w, r_y)$
```

```
Prove$_{\mathsf{crs}}$ $\left((a_k)_{k \in I_{\text{mid}} \cup I_{\text{io}}}\right)$

$\delta_v, \delta_w, \delta_y \leftarrow R^*$
$h'(x) := h(x) + \delta_v w(x) + \delta_w v(x) + \delta_v \delta_w t(x) - \delta_y$
$A := \mathsf{E}\left(v_{\text{mid}}(s) + \delta_v t(s)\right) = \sum_{k \in I_{\text{mid}}} \sum_{i=1}^d a_k v_{k,i} \mathsf{E}\left(s^i\right) + \sum_{i=0}^d \delta_v t_i \mathsf{E}\left(s^i\right)$
$\widehat{A} := \mathsf{E}\left(\alpha v_{\text{mid}}(s) + \delta_v t(s)\right) = \sum_{k \in I_{\text{mid}}} \sum_{i=1}^d a_k v_{k,i} \mathsf{E}\left(\alpha s^i\right) + \sum_{i=0}^d \delta_v t_i \mathsf{E}\left(s^i\right)$
$B := \mathsf{E}\left(w_{\text{mid}}(s) + \delta_w t(s)\right) = \sum_{k \in I_{\text{mid}}} \sum_{i=1}^d a_k w_{k,i} \mathsf{E}\left(s^i\right) + \sum_{i=0}^d \delta_w t_i \mathsf{E}\left(s^i\right)$
$\widehat{B} := \mathsf{E}\left(\alpha w_{\text{mid}}(s) + \delta_w t(s)\right) = \sum_{k \in I_{\text{mid}}} \sum_{i=1}^d a_k w_{k,i} \mathsf{E}\left(\alpha s^i\right) + \sum_{i=0}^d \delta_w t_i \mathsf{E}\left(s^i\right)$
$C := \mathsf{E}\left(y_{\text{mid}}(s) + \delta_y t(s)\right) = \sum_{k \in I_{\text{mid}}} \sum_{i=1}^d a_k y_{k,i} \mathsf{E}\left(s^i\right) + \sum_{i=0}^d \delta_y t_i \mathsf{E}\left(s^i\right)$
$\widehat{C} := \mathsf{E}\left(\alpha y_{\text{mid}}(s) + \delta_y t(s)\right) = \sum_{k \in I_{\text{mid}}} \sum_{i=1}^d a_k y_{k,i} \mathsf{E}\left(\alpha s^i\right) + \sum_{i=0}^d \delta_y t_i \mathsf{E}\left(s^i\right)$
$D := \mathsf{E}\left(h'(s)\right) = \sum_{i=0}^{d-1} h'_i \mathsf{E}\left(s^i\right)$
$\widehat{D} := \mathsf{E}\left(\alpha h'(s)\right) = \sum_{i=0}^{d-1} h'_i \mathsf{E}\left(\alpha s^i\right)$
$F := \mathsf{E}\left(\beta(r_v v v_{\text{mid}}(s) + r_w w_{\text{mid}}(s) + r_y w_{\text{mid}}(s))\right)$
    $= \sum_{k \in I_{\text{mid}}} \mathsf{E}\left(\beta(r_v v_k(s) + r_w w_k(s) + r_y y_k(s))\right)$
return $\pi := \left(A, \widehat{A}, B, \widehat{B}, C, \widehat{C}, D, \widehat{D}, F\right)$
```

```
Verify$_{\mathsf{vk}}$ $\left((a_k)_{k \in I_{\text{io}}}, \pi\right)$

parse $\pi$ as $\left(A, \widehat{A}, B, \widehat{B}, C, \widehat{C}, D, \widehat{D}, F\right)$
$v_{\text{mid},s} := \mathsf{E}^{-1}(A);\ v_{\text{mid},\alpha s} := \mathsf{E}^{-1}(\widehat{A})$
$w_{\text{mid},s} := \mathsf{E}^{-1}(B);\ w_{\text{mid},\alpha s} := \mathsf{E}^{-1}(\widehat{B})$
$y_{\text{mid},s} := \mathsf{E}^{-1}(C);\ y_{\text{mid},\alpha s} := \mathsf{E}^{-1}(\widehat{C})$
$h_s := \mathsf{E}^{-1}(D);\ h_{\alpha s} = \mathsf{E}^{-1}(\widehat{D})$
$l_\beta := \mathsf{E}^{-1}(F);\ l := r_v v_{\text{mid},s} + r_w w_{\text{mid},s} + r_y y_{\text{mid},s}$
$v_{\text{io},s} := \sum_{k \in I_{\text{io}}} a_k v_k(s);\ w_{\text{io},s} := \sum_{k \in I_{\text{io}}} a_k w_k(s);\ y_{\text{io},s} := \sum_{k \in I_{\text{io}}} a_k y_k(s)$
$p := (v_{\text{io},s} + v_{\text{mid},s})(w_{\text{io},s} + w_{\text{mid},s}) - (y_{\text{io},s} + y_{\text{mid},s})$
return $(v_{\text{mid},\alpha s} = \alpha v_{\text{mid},s}) \wedge (w_{\text{mid},\alpha s} = \alpha w_{\text{mid},s}) \wedge (y_{\text{mid},\alpha s} = \alpha y_{\text{mid},s})$
        $\wedge (h_{\alpha s} = \alpha h_s) \wedge (l_\beta = \beta l) \wedge (p = h_s t(s))$
```

Figure 4.1: The Rinocchio protocol (adapted from [GNSV21]). Components needed for zero-knowledge are shown in blue.

We provide the first analysis of the concrete costs of the Rinocchio protocol. In Table 4.1, we present the cost of setting up, proving, and verifying a circuit using Rinocchio, expressed in units of operations over three different rings. This allows for a fair comparison with the cost of the FHE computation to be proven, which operates on the same rings. In particular, this comparison can be made independently of any efficiency improvements to the underlying ring arithmetic library.

Rinocchio operates on three separate arithmetic spaces:

R**:** the base ring, which is also the ring used for the computation to be proven. For FHE, $R = \mathbb{Z}_q[X]/\langle X^N + 1\rangle$, where $q = \prod_{i=1}^{l} q_i$ is a product of l primes, and N is a power of two;

A**:** R's *exceptional set*. For R as above, $A = \mathbb{Z}_{q_1}$, and each element of A can be embedded in R as a constant polynomial;

C**:** the *encoding space*, to which the linear-only encoding $\mathsf{E} : R \to C$ maps. In practice, C will be much larger than R, and operations involving elements of C will be slower than the corresponding operations on R.

Algorithm	$A \times R$	$R + R$	$R \times R$	$C + C$	$R \times C$	$\mathsf{E}(\cdot)$	$\mathsf{E}^{-1}(\cdot)$
Compute		$\leq 2^d - m - 1$	$m + 1$				
Prove	$6dm$	$6d(m-1)$		$m + 8d - 9$	$8d$		
Setup	$3m + d$	$2m$	m			$m + 2d + 2$	
Verify	$\leq 3l + 1^\dagger$	$\leq 3l + 3^\dagger$	9				9

†: for layered circuits (as often used in practice), v, w, and y will be sparse, and the costs here will be $l + 1$ operations instead of $3l + 1$ (respectively $l + 3$ instead of $3l - 3$)

Table 4.1: Costs (expressed as operations over A, R, C) of the Rinocchio protocol (without zero-knowledge), for $l = |I_{\mathsf{io}}|$ inputs and outputs and $m = |I_{\mathsf{mid}}|$ intermediate/private values.

4.1.2 Improving Encoding Efficiency

In [GNSV21], Ganesh et al. introduce two possible encodings for the cyclotomic rings used by FHE. The first one (dubbed "Regev-style" encoding) encodes each of the N coefficients in $\mathbb{Z}_q$ by encrypting it into an element of $\mathbb{Z}_Q^n$ using a LWE cryptosystem scheme; the parameters of the encoding scheme are chosen to ensure that the encodings are k-linearly-homomorphic, where k is determined by the circuit. The second construction ("Torus encoding") uses a variant of the TFHE cryptosystem.

The Regev encoding has an expansion factor of $\frac{N \cdot n \cdot \log_2(Q)}{N \cdot \log_2(q)} = n \cdot \log_q(Q)$, as it encodes each of the N coefficients in $\mathbb{Z}_q$ as an element of $\mathbb{Z}_Q^n$. However, most FHE implementations will not be able to support a plaintext modulus of the size of q (typically hundreds of bits), and in practice one would need to encode each of the l CRT components individually, leading to an expansion factor of $l \cdot n \cdot \log_q(Q)$. Using this encoding will thus slow down the prover and verifier significantly, as all encodings, decodings, and computations over the encoding space will be slow.

The TFHE encoding, on the other hand, requires using floating-point arithmetic to encode and decode, unlike the FHE scheme used for computation. Additionally, this encoding does not allow us to use the (potentially heavily optimised) ring arithmetic libraries provided by the FHE library.

Therefore, we propose a new RLWE Regev-style encoding for Rinocchio, taking advantage of the batching technique commonly used in FHE. For many FHE schemes, if the plaintext modulus t satisfies the condition $t = 1 \mod 2N$, one can use an efficient encryption that packs N plaintext values (interpreted as an element of R_t) into a single ciphertext in R_q^2. For our encoding, we take an input in R_q as l polynomials in $R_{q_1}, \dots, R_{q_l}$ (this decomposition is already used natively by the FHE scheme for efficiency reasons), and encode each of those polynomials as an element in R_Q. The expansion factor in this case is $\frac{l \cdot \log_2(Q)}{\log_2(q)} = l \cdot \log_q(Q)$, improving on the Regev encoding by a factor of $N \geq 2^{10}$. Using this batching technique imposes the requirement $q_i = 1 \mod 2N$ on the ciphertext moduli of the FHE scheme; this condition is already necessary for some schemes (e.g., RNS-optimised BGV [KPZ21]), and can be easily satisfied for all other schemes.

Table 4.2 shows the asymptotic complexity of ring operations when using this new encoding.

Operation	$A \times R$	$R + R$	$R \times R$	$C + C$	$R \times C$	$\mathsf{E}(\cdot)$	$\mathsf{E}^{-1}(\cdot)$
$\Theta(\cdot)$	Nl	Nl	$Nl \log(Nl)$	NL	$NL \log(NL)$	$NL \log(NL)$	$NL \log(NL)$

Table 4.2: Asymptotic complexity of ring operations (for the batched Regev encoding with $Q = \prod_{i=1}^{L} Q_i$, see Section 4.1.2)

4.1.3 Implementation & Evaluation

We implemented a prototype of Rinocchio as a template-based C++ library [Knab], which includes the core protocol, and provides an interface for potential future implementations of efficient encoding functions. Our library implements Rinocchio generically (such that the protocol can be used with any FHE library, a user is only required to provide a backend for the underlying ring arithmetic), together with a concrete instantiation of the protocol using the SEAL [Sea] library (version 4.0). For this integration with SEAL, our implementation relies on the polytools [VK] library, which we extended as part of this thesis.

Our implementation has been used by researchers at Intel Labs to investigate hardware-acceleration of Rinocchio, and we plan to open-source our library shortly after the publication of this thesis.

By relying on an existing and widely deployed FHE library, we allow practitioners to add integrity protection to their FHE deployments with a minimal amount of additional dependencies, and we ensure a seamless integration with the FHE circuit to be proven. Additionally, our implementation of Rinocchio immediately benefits from any efficiency improvement of the underlying ring operations; for example, our implementation can be hardware-accelerated using the Intel Homomorphic Encryption Acceleration Library (HEXL) [Boe+21] (which is a supported backend for SEAL) by simply compiling SEAL with the corresponding flag enabled.

4.1.4 Discussion

Despite its native support for FHE-friendly rings, Rinocchio suffers from limitations that prevent it from being usable for FHE as-is. First, its expressiveness is limited, as Rinocchio only supports arithmetic circuits over rings, whereas some FHE operations (e.g., relinearization) use component-wise rounding operations internally. Rinocchio also only supports a single ring for a given circuit, whereas some FHE schemes (e.g., B/FV) use as much as three rings for multiplication [FV12]; while this issue can be circumvented by using the biggest ring throughout, and emulating modulus switching to smaller rings, it still worsens Rinocchio's performance.

Additionally, the soundness of Rinocchio relies on the size of the exceptional set, which for FHE-friendly cyclotomic rings is $|A| = q_1 \approx 2^{60}$, i.e., Rinocchio provides around 60 bits of (computational) soundness; in practice, one would need to use a soundness amplification strategy to achieve a more satisfactory level of soundness. Finally, Rinocchio is designated-verifier, which may limit its applicability as a building block in a construction for some applications.

Rinocchio is thus much more FHE-friendly than previous proof or argument systems, but still struggles to efficiently represent state-of-the-art FHE optimizations natively. Rinocchio is also moderately flexible, but can be coupled very closely to an efficient FHE implementation to directly benefit from its optimization.

4.2 Hardware-based Primitives

TEEs are hardware components capable of isolating code running on them from the rest of the operating system. Code running on a TEE is free from tampering from other processes, even from the operating system or hypervisor. TEEs are commercially available in the form of commodity hardware, provided by all major hardware vendors (e.g., Intel SGX [Sgx], ARM TrustZone [Gea15], AMD SEV [Amd]).

4.2.1 Analysing Complexity & Expressiveness

In the security and privacy literature, TEEs have been used to guarantee two distinct properties: integrity and confidentiality Integrity is guaranteed by the isolation of the TEE from the rest of the operating system, and a TEE can prove to other parties that it executed a specific signed binary by producing a *remote attestation*. Despite a few attacks [Mur+20], TEEs are generally accepted to provide integrity. Confidentiality, on the other hand, is seldom guaranteed only by the TEE, and usually requires additional hardening by the TEE user. In particular, most TEEs are very vulnerable to side-channel attacks, which leak information about data within the enclave to an outside adversary. A slew of side-channels attacks on TEE [NBB20] point out the brittleness of their privacy guarantees of TEEs, which has spurred a new line of research combining FHE (for privacy) and TEE (for integrity).

This combined approach [DG17; Bre+22; Nat+21] makes use of TEEs for their integrity properties, by running an entire existing FHE pipelines inside an enclave. This approach also allows for more complex setups with additional parties, providing different privacy and integrity guarantees for each party. Since all these works simply embed a FHE pipeline within a TEE, they are restricted by the enclave's hardware limitations, which means that they suffer from heavy performance degradation when a lot of working memory is required (which is a concern due to FHE's ciphertext expansion).

Some works using TEE use them as fast secure computation units [TB18; BMA20]. These works rely heavily on TEEs for privacy, and thus suffer from the restrictions and concerns outlined above. Other works still [Cop+21; Wan+19; Sad+19] have explored the use of TEEs as (fast) private computation modules, but they do not aim to provide computation integrity, and are thus of limited interest for this thesis.

In this thesis, we do not rely on any data confidentiality guarantees that a TEE might provide, and instead rely on the much more widely accepted integrity guarantees[2]. In particular, we assume that the computing party has access to a TEE. We follow the *transparent enclave* model from Tramèr et al. [Tra+17], which captures this intuition. Conceptually, for a relation $\mathcal{R} = \{(x, w) \mid f(x, w) = 1\}$ with an efficiently computable f, one can construct a zero-knowledge proof protocol as follows: First, the prover and verifier reach

[2]Technically, the integrity of TEEs hinges on the secrecy of some cryptographic material, and we do rely on these (signing and attestation) keys remaining secret. However, these are more protected than the data loaded on the TEE, and require much fewer trust assumptions, as a much smaller portion of the TEE design and implementation needs to be trusted.

an agreement on a program that implements f, but does otherwise not output any other information (e.g., does not leak w). The verifier then loads this program together with its input x on the prover's TEE. The prover runs this program with its own private input w, and attests to the verifier that the program was run correctly using the partial input x, thereby convincing the verifier that x is in the language corresponding to $\mathcal{R}$. We note here that we assume that the verifier uses a separate machine from the prover (i.e., it cannot recover w by using side-channel attacks), as would be the case in our setting (where the client and server would not be on the same machine). For more details, we refer the interested reader to the paper by Tramèr et al. [Tra+17].

One can see immediately that a TEEs are very flexible primitives, since they rely on a very generic underlying representation; indeed, any relation that can be expressed as a program can be proven. In particular, TEEs can cope very well with non-arithmetic sub-operations in FHE operations (e.g., rounding).

TEEs are also relatively efficient, but only up to a certain point. The only work (to the best of our knowledge) that investigates running FHE workloads on TEEs [Nat+21, Table 4] report a $\times 2.4$ slowdown compared to a normal CPU, which is consistent with our measurements (see Section 4.2.3). However, as TEEs encrypt/decrypt memory on paging, paging data in and out of an enclave incurs a heavy latency penalty [Gje+17]. Additionally, TEEs have smaller physical memory limits than conventional CPUs, which means that application requiring large amounts of memory incur significant overhead when running on TEEs.

4.2.2 Accelerating FHE-in-TEE with Untrusted Hardware

Executing any code inside a TEE incurs a slowdown (due to reduced computational power and memory), especially in the case of FHE computations, that are typically compute- and memory-intensive. To alleviate this slowdown, we propose a new method to accelerate FHE computations inside TEEs by taking advantage of faster (but untrusted) hardware (e.g., a vanilla untrusted CPU, a CPU with specialised vector instructions repurposed for FHE [Boe+21], a GPU accelerator for FHE [Oze+21], or even a dedicated hardware accelerator).

The key insight to our improvement is that both the TEE and the untrusted hardware are on the side of the (malicious) server. Therefore, the server's input does not need to be protected from the server, and can be stored on the server's own untrusted hardware; the client's inputs are only available in their encrypted form, and can thus also be stored outside the enclave. This insight allows us to devise a protocol for *verifiably* outsourcing certain FHE operations. In order to do this efficiently, we rely on a very lightweight, information-theoretic argument of equality, based on the generalised Schwartz–Zippel lemma over rings:

Theorem 4.1 (Generalised Schwartz–Zippel Lemma over Rings [Bis+18; GNSV21]) *For a ring R, let $f : R^n \to R$ be a n-variate non-zero polynomial, let $A \subseteq R$ be a finite exceptional set, and let* $\deg(f)$ *denote the total degree of f. Then:*

$$\Pr_{\vec{a} \leftarrow A^n}[f(\vec{a}) = 0] \leq \frac{\deg(f)}{|A|}$$

For a given computation, we encode the expected result in one polynomial (f), and the actual result computed on untrusted hardware in another polynomial (g). The trick for efficiency, then, is to compute the compute and compare $f(a)$ and $g(a)$ faster than computing the full representation of g in the first place.

Consider, for example, the *tensoring* operation, which is the most computationally expensive part of FHE multiplication (for the B/FV, BGV, and CKKS schemes). In the following, we will interpret a ciphertext $\mathbf{ct} = (\mathbf{ct}_0, \dots, \mathbf{ct}_{k-1}) \in R_q^k$ as a polynomial of degree $k-1$ over R_q, where $\mathbf{ct}_i$ is the i-th coefficient.

The tensoring operation takes as input two ciphertexts $\mathbf{ct} = (\mathbf{ct}_0, \mathbf{ct}_1), \mathbf{ct}' = (\mathbf{ct}'_0, \mathbf{ct}'_1) \in R_q^2$, and outputs $c_{\text{out}} = \mathbf{ct} \cdot \mathbf{ct}' = (\mathbf{ct}_0 \cdot \mathbf{ct}'_0, \mathbf{ct}_0 \cdot \mathbf{ct}'_1 + \mathbf{ct}'_0 \cdot \mathbf{ct}_1, \mathbf{ct}_1 \cdot \mathbf{ct}'_1) \in R_q^3$. Now, evaluating the expected result $\mathbf{ct} \cdot \mathbf{ct}'$ at a random point $a \in A$ can be done efficiently as follows:

$$f(a) := (\mathbf{ct} \cdot \mathbf{ct}')(a) = \mathbf{ct}(a) \cdot \mathbf{ct}'(a) = (\mathbf{ct}_0 + a \cdot \mathbf{ct}_1) \cdot (\mathbf{ct}'_0 + a \cdot \mathbf{ct}'_1)$$

Evaluating the untrusted result $\mathbf{ct}_{\text{out}}$ at this same point can be done by using Horner's rule:

$$g(a) := \mathbf{ct}''(a) = \mathbf{ct}''_0 + a \cdot (\mathbf{ct}''_1 + a \cdot \mathbf{ct}''_2)$$

After checking that $f(a) = g(a)$, we know that $\mathbf{ct} \cdot \mathbf{ct}' = \mathbf{ct}''$ with high probability (for $R = \mathbb{Z}_{q_1 \cdot \ldots \cdot q_l}[X]/f[X]$, we have $|A| = q_1 \approx 2^{60}$, i.e., 60 bits of soundness). While computing the result has a concrete complexity of 1 $_R{+}_R$, 4 $_R{\times}_R$, verifying the result as outlined above only requires 4 $_A{\times}_R$, 4 $_R{+}_R$, 1 $_R{\times}_R$.

This approach can also be extended as follows to verify k tensoring operations at the same time. Let

$$f(a_1, \dots, a_k) := \sum_{i=1}^{k} (\mathbf{ct}_i \cdot \mathbf{ct}'_i)(a_i) = \sum_{i=1}^{k} (\mathbf{ct}_{i,0} + a_i \cdot \mathbf{ct}_{i,1}) \cdot (\mathbf{ct}_{i,0} + a_i \cdot \mathbf{ct}'_{i,1}),$$

and define

$$g(a_1, \dots, a_k) := \sum_{i=1}^{k} \mathbf{ct}''_i(a_i) = \sum_{i=1}^{k} (\mathbf{ct}''_{i,0} + a_i \cdot (\mathbf{ct}''_{i,1} + a_i \cdot \mathbf{ct}''_{i,2})).$$

Computing k tensoring operations has a concrete complexity of

$$k \ {}_{R}{+}_{R}, 4k \ {}_{R}{\times}_{R},$$

while verifying the result by computing $f(\vec{a})$ and $g(\vec{a})$ has a complexity of

$$4k \ {}_{A}{\times}_{R}, (6k-2) \ {}_{R}{+}_{R}, k \ {}_{R} \times_{R} .$$

By trading expensive R-R multiplications for cheaper R-R additions and A-R multiplications, we are able to achieve a non-negligible speed-up, which we quantify in the next section.

We can view our protocol as a much more efficient, non-zero-knowledge version of Rinocchio; indeed, Rinocchio also uses Theorem 4.1, but requires significantly more protocol machinery in order to achieve zero-knowledge. In addition, Rinocchio offers roughly $\log_2(q_1) \approx 60$ bits of computational soundness (and thus requires a soundness amplification strategy), while our protocol offers $\log_2(q_1)$ bits of *statistical soundness*, and can therefore provide a satisfactory level of security by itself.

We note that this optimization is similar to the Slalom framework by Tramèr and Boneh [TB18], which offloads matrix multiplications (pover unencrypted) values to untrusted hardware by using Freivalds' algorithm. Slalom relies on the TEE both for integrity and data confidentiality and only supports matrix multiplication, whereas our protocol does not require confidentiality of the data stored on the TEE, and can handle arbitrary polynomial computations.

4.2.3 Implementation & Evaluation

We implemented our framework [Knaa] based on the SEAL library [Sea] and the OpenEnclave SDK [Oe], which allows practitioners to quickly port existing FHE applications onto the TEE backends supported by OpenEnclave (including Intel SGX). We also implemented outsourcing and verification procedures for FHE as described above, which can be used as faster replacements for operations in existing FHE applications.

To the best of our knowledge, the only other work leveraging FHE and TEEs to achieve integrity is CHEX-MIX [Nat+21], which provides a high-level description on making the SEAL library run on Intel SGX, but does not provide an open-source implementation, and does not make use of the computational power of the untrusted hardware to accelerate the computation.

Table 4.3 shows benchmarks for our verification algorithm on Intel SGX. We first note that multiplication on untrusted hardware (without any further optimizations) is roughly 2 times faster than on the TEE (on our machine), which is consistent with previous reports from the literature [Nat+21]. The cost of transferring ciphertexts in and out of the enclave ("Setup") are on the order of ($\approx 10^4\,\mu$s), and grows very moderately as the number of offloaded ciphertexts increases (for 32 ciphertexts, being 2 orders of magnitude faster than either computing on untrusted hardware or verifying on the TEE).

Verifying the result of a multiplication is consistently $\times 1.4$–2.5 times faster than performing the computation inside the TEE ($\times 1.8$–2.5 for batched verification); batched verification offers a speedup of around 20% compared to the non-batched version (starting at $k = 8$ already).

For any FHE application, we can thus offload a computation to the untrusted hardware while verifying the result of the previous computations inside the TEE. By streamlining computation in this manner, practitioners get a hardware-backed guarantee for the integrity of their FHE computation at a negligible cost compared to an unprotected run.

k	Multiplication (TEE)	Multiplication (Untrusted)	Verification (TEE)	Batched Verif. (TEE)	Setup (TEE $\leftrightarrow$ Untr.)
1	7.40×10^4	3.52×10^4	2.91×10^4	2.93×10^4	1.08×10^4
8	5.91×10^5	2.85×10^5	4.15×10^5	3.21×10^5	1.37×10^4
32	2.36×10^6	1.15×10^6	1.64×10^6	1.30×10^6	2.30×10^4
45	3.34×10^6	1.61×10^6	2.28×10^6	1.82×10^6	2.79×10^4

Runtimes in μs, averaged over 5 runs. FHE parameters: $N = 2^{12}$, BGV scheme with a 109-bit q (3 moduli, SEAL default parameters for $\kappa = 128$). Untrusted hardware: Intel Core i5 CPU (1.60GHz). TEE: Intel SGX.

Table 4.3: Runtimes of k sequential multiplications on a TEE versus runtimes of multiplications on untrusted hardware with verification on a TEE.

4.2.4 Discussion

Due to their very expressive internal representation, TEEs are very FHE-friendly and flexible (the only hurdle is porting an FHE library to work with the reduced standard library available on the TEE). Contrastingly, using TEEs by themselves does not allow us to take advantage of improvements in FHE libraries. This is remediated by our FHE-in-TEE optimization, which allows us to only perform verifications inside the TEE, while offloading heavy computations to faster hardware.

While there have been significant improvements in the space of integrity primitives for FHE (both cryptography-based and hardware-based), existing primitives still suffer from a lack of expressiveness and/or efficiency, which prevents them from being of practical use at the moment. In addition, these integrity primitives are only used to prove that a certain FHE circuit is satisfied (see Chapter 3). As we will see in Chapter 5, this does not offer enough guarantees in the presence of a malicious adversary for realistic FHE settings.

Chapter 5

Realistic Security Notions for FHE

As studied in Chapter 3, there is a variety of FHE approaches with different security and functional guarantees. In this chapter, we first analyse the requirements of real-world FHE applications, which we compare and contrast them with the guarantees given by existing approaches. We show that prior work falls woefully short of achieving these properties. We also show concrete attacks against the security of each of these deployment scenarios, even if they are protected by state-of-the-art verifiable computation schemes. We also show two complete key-recovery attacks that demonstrate the shortcomings of existing integrity frameworks for FHE. We formulate these attacks on the level of generic constructions, showing that these are not issues with specific instantiations, but rather a fundamental mismatch between the requirements of FHE deployments and the notions of integrity supported by state-of-the-art approaches. Finally, we formally define stronger and more expressive integrity notions for FHE, which naturally generalise existing definitions, and show that schemes that satisfy these new notions are secure against our attacks.

5.1 FHE Applications in Practice

FHE applications can be broadly categorised by the amount of hidden inputs they require. Firstly, an application can have inputs only from one party (e.g., a client provides all ciphertexts and a public circuit, and asks a more powerful server to evaluate this circuit on this input), and we will study such applications in Section 5.1.1. For a second class of applications, some inputs may be private; either the circuit could be public, but some inputs would be provided by the server, or all inputs may be public, but the server applies a secret circuit to them, or both some inputs and the circuit could be unknown to the client. Such applications will be studied in Section 5.1.2.

5.1.1 Outsourced Computation

In the outsourced computation setting, a client is in possession of its (private) inputs and a circuit F, but it wants to offload the computation of this circuit to a more powerful server. To ensure privacy, the client first encrypts its data using FHE, and sends its ciphertext together with the circuit F to the server.

This setting is the most studied one in the VC literature and for FHE integrity (see Chapter 3 for a more complete overview). However, outsourced computations only covers a tiny subset of FHE use cases, due to the slowdown and limitations introduced by FHE. Indeed, in an overwhelming majority of cases, it is much faster to compute over plaintexts than to encrypt and send all inputs, wait for the answer from the server, and to then decrypt (and potentially verify) this result. By using FHE, the client also needs to make sure that the computation can be expressed as an arithmetic circuit over $\mathbb{Z}_t$, and forfeits the use of conditional control flow and loops, which reduces the set of useful applications even further. For this reason, most systems using FHE are *multi-party systems*, and take advantage of FHE to achieve privacy of a party's input with respect to other parties.

5.1.2 Secure Function Evaluation

Most FHE applications in practice are concerned with the evaluation of a circuit on inputs provided by different parties. This computation can be performed by one party (e.g., the server in a typical client-server setting), or by each party independently (e.g., for federated learning). Before studying the biggest families of use cases for such multi-party FHE deployments in the following paragraphs, we would first like to highlight a particularity of the guarantees given by the VC literature (see Chapter 3): these approaches guarantee circuit satisfiability, i.e., they guarantee that for the circuit f and for the client input x, there exists a w such that (after verification and decoding) $y = f(x, w)$. We show how to exploit this lack of guarantees about w to mount a variety of application-specific attacks.

Private Information Retrieval (PIR)

(Single-server) PIR is typically a two-party application, and allows a client to retrieve an item from a server's database, without revealing which item was retrieved. The database can either be private to the server, or it could have sent by the client to the server during a one-time setup phase. The most efficient (single-server) PIR protocols are built using FHE [Ang+18; GH19; PT20; Ali+21; MCR21], following Kushilevitz and Ostrovsky's initial construction to realise PIR with a HE scheme [KO97].

Because the client will not have access to the full database at the time of making a request (either because the database is private to the server, or because the client offloaded it first), the client typically knows only the ciphertext encrypting its query, while the server knows both the client's encrypted query and the database. The circuit used to privately retrieve a record is public and does typically not depend on a particular use case, but follows one of the state-of-the-art approaches.

For PIR, the entire database must be scanned and (homomorphically and privately) compared to the client's query in order to preserve privacy; this makes PIR computationally expensive in practice, and a malicious (even rational) server would thus have an incentive to save some of this computation. For example, by choosing its input to circuit after receiving the client's query, a server could make it appear to the client as if the server database consisted only of the client's query, or it could claim to have a reduced (or even empty) database in order to save some computation. The server could also use different databases for different user queries, whereas the user would expect the server to use the same database for an extended period of time. Both of these attacks could be thwarted either by having the server (privately) committing to its database at the beginning of the protocol, or by the server proving that an amount of computation equivalent to that needed to match a query against a database of a certain size; in the following, we will call such a mechanism a *Proof-of-Computation*[1]. The latter attack can be thwarted using the commitment mechanism as well.

Private Set Intersection (PSI) and Private Set Union (PSU)

In PSI/PSU, each party has a set of records, and the protocol allows a party to learn the intersection or the union of its set with that of the other parties. For this use case, some of the most efficient systems rely on FHE (especially in the asymmetric case, where one set is much larger than the other) [CLR17; Che+18; Con+21; Tu+22].

PSI/PSU applications have very similar characteristics as PIR applications. Firstly, they usually involve only two parties, for example as in the Microsoft Edge Password Monitor [Lau+21], where the client's set is composed of a user's saved passwords in their browser, and the server set is a database of compromised credentials. Similarly, there is usually only one circuit, which is publicly known. Finally, the attacks against PIR with a malicious server have a direct analogue for PSI/PSU applications, and the same countermeasures (input commitment and proof-of-computation).

[1]"Computation" here is to be understood as *computational effort*; we refrain from using terms related to the semantically-loaded "Proof-of-Work" as used in the blockchain community.

Machine Learning (ML)

Privacy-preserving ML is a major application for FHE in practice, for example in the biomedical sector [Wan+18]. The setup and characteristics of the system depend on whether the application aims to evaluate an existing model (inference), or whether it aims to train a new one. However, ML applications make heavy use of approximate FHE, as both ML inference and training can be implemented much more efficiently using floating-point representations.

Inference. In this type of system, the server holds a ML model, and the client holds several private inputs. The server usually wants its model to remain private to preserve the intellectual property of the model owner, and to prevent the model from being stolen. The client's input is also private, and is sent in encrypted form to the server. FHE has been used for ML inference for a wide variety of model architectures and use cases [AEH15; GB+16; WNK20; Iez20; Con+22; Lee+22].

We see here again that ML inference is a two-party setting, with private inputs from both parties. The circuit that evaluates the model may be either public, if the model owner is willing to disclose the topology of its model, or private otherwise. However, the client still needs to receive some information about the depth of the network (in order to use the right FHE parameters), and the topology of a network by itself usually discloses little information about the model itself, which is primarily determined by the exact model weights. For these reasons, the circuit is usually assumed to be public in the literature, and there is only circuit per model that is evaluated repeatedly for different client inputs.

A malicious server could try to save on computation by claiming to evaluate a deeper and more complex model than what it actually evaluates (and bill the client accordingly); such an attack would be defeated either by input commitment or by providing a proof-of-computation. The server could also use different machine learning models for different queries, which would be prevented using input commitments.

Training. Related to ML inference, ML training is often achieved by *federated learning*; several parties each hold their private datasets, and they collaboratively train a join model incrementally. Typically, in these systems each party performs some FHE computations on its private data, before exchanging the result with other parties (either through a central aggregation server, which also evaluates a circuit, or by using a MPC protocol). FHE has been used extensively for ML training for a wide variety of models and applications [Kim+18; WNK20; Iez20].

In this setting, the circuit is public and reused at every iteration of the training, and each party needs to cope with private inputs from other parties.

Similarly to the ML inference setting, a malicious adversary could misrepresent the amount of computation it has performed, or it could switch out its training dataset in different iterations; these attacks would be prevented by using input commitment or proofs-of-effort, respectively by using input commitment. Another attack is enabled by the lack of guarantees on the circuit's inputs: by using inputs with disproportionate sizes, a malicious participant could poison the model being trained, preventing it from converging, or forcing it to perform poorly for some inputs. Existing VC schemes only provide a very coarse check, guaranteeing that server inputs lie in R_q, which is not very useful in practice.

Generic Secure Function Evaluation

FHE is also used in a variety of multi-party use-cases to protect the privacy of each party's inputs. These applications range from Vehicular Ad-Hoc Networks (VANETs) [Sun+20], securing drone systems [SAK17] and aerial photography [SAK17], for biometric data processing [TBS14; ABS15], and for the defence and intelligence sector [Geu18]. The exact characteristics for other multi-party FHE use cases that do not fit our classification above depend on the specific setting; however, we believe that all the guarantees outlined above are generally desirable to achieve satisfactory integrity guarantees for a wide variety of use cases.

5.1.3 Characteristics and Desirable Guarantees for Real-World FHE

Table 5.1 summarises the most common families of FHE applications, well-known attacks against the correctness of the application, and guarantees that would prevent these attacks.

We first see that most FHE applications are multi-party (and usually two-party), and that they mostly use a single public circuit (or few of them), which are evaluated repeatedly on different inputs. Hence, circuit privacy is very often not required, but support for context-hiding is paramount. We also note that the usage of a single circuit may make an amortised VC scheme viable, as an expensive pre-processing phase for one circuit can be amortised over many repeated evaluations (besides, FHE by itself also requires a somewhat expensive pre-processing phase, due to the generation of public and evaluation keys, and the potential first encryption of the server's inputs).

These setups also question the need for *public delegatability* and *public verifiability* for many FHE use cases: both notions already requires a lot of trust in the setup-party (which generates the secret keying material), and delegators won't be able to decrypt result by themselves. We note that allowing for (variants of VC schemes) without support for public delegatability and/or verifiability may lead to more efficient primitives, without compromising the

Application	Cont.-Hiding	Func. Priv.	Multi-Circ.	Inp. Com.	PoC	Well-Form.	Appr. Comp.
Outsourced Computation			●				●
Private Information Retrieval	●			●	●	●	
Private Set Intersection/Union	●			●	●	●	
Machine Learning Inference	●			●	●	●	●
Machine Learning Training	●			●	●	●	●
Secure Function Evaluation	●	●	●	●	●	●	●

Cont.-Hiding : context-hiding is desirable, i.e., an adversary does not learn additional information about the function's inputs from its outputs, even if it holds the secret key;

Func. Priv. : function privacy is desirable, i.e., the client does not learn the function f;

Multi-Circ. : the application is multi-circuit, i.e., it routinely switches between many different circuits, instead of using a few circuits throughout;

Inp. Com.: the application benefits from having other parties have to commit to their input w;

PoC: the computing party provides a proof-of-computation, i.e., it proves that it expended a computational effort comparable to the effort needed to evaluate the circuit;

Well-Form.: the application benefits from well-formedness checks on inputs;

Appr. Comp.: the application benefits from using approximate computation.

Table 5.1: Characteristics and desirable guarantees for real-world FHE applications.

utility of a broad class of FHE applications. Therefore, we focus on privately verifiable VC schemes when sketching more practical constructions in Chapter 8, but we present our generic constructions and prove their security generally, without choosing public or private verifiability.

Many attacks against the correctness of FHE applications can be thwarted by using *input commitments*, *proofs-of-effort*, and *well-formedness* checks on inputs. *Approximate* FHE schemes are also used often in practice, and their support is therefore a desirable property for a VC scheme. Finally, we note that FHE is (due to its round and communication-efficiency) is usually used in low-interaction environments (with plenty of opportunities to batch multiple messages together); a desirable and realistic and FHE-friendly VC scheme would therefore be a non-interactive as possible, and succinct. In the following section, we will now show that unfortunately, existing integrity notions do not cover these desirable guarantees.

5.2 Shortcomings of Existing Notions

In the previous section, we have shown that realistic FHE applications are vulnerable to attacks on their correctness, even when protected with state-of-the-art verifiable computation. However, the gap between theory and practice in existing FHE integrity approaches is not only devastating for correctness, but is also directly detrimental for privacy. In this section, we show two complete key-recovery attacks against a generic FHE scheme protected with state-of-the-art VC against a realistic adversary that is slightly stronger than what was considered in the VC literature. In particular, these attacks are black-box with respect to the FHE and VC scheme, as long as the FHE scheme is only IND-CPA-secure (which is the case for all modern FHE schemes), and as long as the VC scheme is built using the paradigms introduced in Section 3.1.

5.2.1 Key-Recovery Attacks against VC-protected FHE

In the FHE literature, attacks against FHE have been known for a long time, indeed since the very first generation of schemes. Already in 2012, Zhang et al. [ZPS12b] showed a complete key recovery attack for Gentry's original scheme in a realistic outsourced computation scenario. In parallel, Loftus et al. [Lof+12] presented key recovery attacks (in the CCA1 setting) against this same scheme. Later, Zhang et al. [ZPS12a] presented another key recovery attack against similar FHE schemes. Chenal and Tang [CT15] continued this line of research by presenting CCA1 key-recovery attacks against newer schemes (BV, BGV, GSW).

Later, Chillotti et al. [CGG16] presented several generic reaction attacks on FHE deployments; in essence, whenever the client reacts after a faulty result received from the server, it provides an oracle that a malicious adversary could exploit to recover information about the client's plaintexts. In particular, when using a classical (unprotected, i.e., only IND-CPA-secure) FHE scheme, a client will have no way to check whether a ciphertext is malformed, and will thus have to first decrypt it. However, if the resulting plaintext has been tampered with in an unexpected manner by the server, the client might react differently, thereby leaking information about the plaintext, but also potentially about the secret key. One common countermeasure outlined by these papers is to use VC schemes in order to guarantee security in a malicious setting (beside not implementing a decryption oracle in the first place, which is a very strong requirement for FHE applications).

However, the strongest adversary model assumed in the VC literature considers adversaries with access to a verification oracle only, whereas in practice, it is likely that an adversary would have access to a *decryption oracle*. Additionally, using the VC schemes proposed in the literature is not enough to protect privacy against these stronger adversaries, as we show in the following two attacks:

Standalone Key-Recovery Attack against VC-protected FHE

We first recall the key recovery attack against the BV scheme from Chenal and Tang [CT15], which we will then trivially extend to attack a VC-protected scheme. In BV, decryption is performed as $\mathsf{Dec}_{\mathsf{sk}}(\mathbf{ct}) = [\mathbf{ct}_0 + \mathbf{ct}_1 \cdot \mathsf{sk}]_t$, where the inner operations are performed over R_q (see Definition 2.2 for details). For a suitable choice of $t > \max_{e \leftarrow \chi}(|e|)$, by trying to decrypt $\mathbf{ct} = (0, 1)$, one trivially recovers the secret key $\mathsf{Dec}_{\mathsf{sk}}(\mathbf{ct}) = [0 + 1 \cdot \mathsf{sk}]_t = [\mathsf{sk}]_t = \mathsf{sk}$. For the case where t does not satisfy the condition outlined above (e.g., $t = 2$), the attack requires additional decryption queries; we refer to [CT15] for the full details.

Attack 1. Key-Recovery Attack against VC-protected FHE (Standalone) Let $\mathcal{VC}$ be any Compute-then-Prove VC scheme as defined in Section 3.1.5 instantiated with the BV cryptosystem (e.g., the non-interactive version of the VC scheme from [Boi+21, Section 3.2]). We need not specify the proof system used for proving, as the attack uses it as a black-box.

By trivially porting the attack against IND-CCA1 from [CT15] to the VC setting, we can construct the following adversary $\mathcal{A}$ against Priv for $\mathcal{VC}$.

```
$\mathcal{A}_1^{\mathsf{KGen},\mathsf{Dec}}(\mathsf{pk})$
$\mathsf{pk}_{\mathsf{id}} \leftarrow \mathcal{O}_{\mathsf{KGen}}(\mathsf{id})$
$c_x := (0, 1)$
$(\sigma_x, \tau_x) := (c_x, c_x)$
$\sigma_s = (c_y, \pi) \leftarrow \mathsf{Compute}_{\mathsf{pk}_{\mathsf{id}}}(\sigma_x)$
$s \leftarrow \mathcal{O}_{\mathsf{Dec}}(\sigma_s, \tau_x)$
$x_0 := 0;\ x_1 := 1$
return $x_0, x_1$
```

```
$\mathcal{A}_2^{\mathsf{KGen}}(\widehat{\sigma}, \widehat{\tau})$
$\widehat{x} := \mathsf{Dec}_s(\widehat{\sigma})$
return $\widehat{x} = x_1$
```

For favourable choices[2] of t, the adversary recovers the secret key as s (and therefore wins the game) if and only if $\mathcal{O}_{\mathsf{Dec}}(\sigma_s, \tau_x)$ does not return $\perp$, i.e., if $\mathsf{Verify}_{\mathsf{pk}_{\mathsf{id}}}(\sigma_s, \tau_x) = 1$. As $\sigma_s = \mathsf{id}(\sigma_x)$, this will indeed be the case by the completeness property of the proof system. We note here that for this scheme, there is no function-specific secret key sk_f, and decryption only requires the FHE secret key sk.

The reason that this attack works is that VC schemes built using the Compute-then-Prove paradigm only guarantee that the output σ_y is the evaluation of the circuit on the input σ_y, but does not offer any guarantees on the well-formedness of the ciphertexts in σ_x. While the choice of the identity function for f makes it trivial to propagate a malformed ciphertext from f's inputs to its output, we note that can be done just as easily for more realistic circuits. For example, consider the circuit $f(x_1, w_2, w_3) := \langle x_1, w_2 \rangle + w_3$ (where the inner product $\langle \cdot, \cdot \rangle$ is computed using one multiplication and repeated rotations

[2]The attack can be extended to handle unfavourable choices of t, at the cost of additional queries to $\mathcal{O}_{\mathsf{Dec}}$, see [CT15] for details.

and additions); such a circuit could for example implements a matrix-vector multiplication. Given a fixed x_1 by the client, the adversary could choose $w_2 = 0$ and w_3 as σ_s above, prove that $f(x_1, w_2, w_3) = \sigma_s$, and recover the secret key by querying its decryption oracle.

Key-Recovery Attacks against VC-protected FHE using Evaluation Keys

Attack 1 uses an obviously invalid ciphertext to recover the private key, which could easily be detected. However, key-recovery attacks can take also take advantage of the *evaluation keys* provided to the server. For FHE, an evaluation key is an encryption of a plaintext related to the secret key under this same secret key, which allows performing ciphertext maintenance operations (e.g., relinearization, key-switching, bootstrapping). Given access to a decryption oracle, an adversary can decrypt one of these evaluation keys, and recover the secret key.

For example, Attack 2 shows such an attack on the BGV scheme protected by a Compute-then-Prove approach.

Attack 2. Key-Recovery Attack against VC-protected FHE (Evaluation Key) Let $\mathcal{VC}$ be a generic Compute-then-Prove VC scheme for BGV (e.g., the non-interactive VC scheme from [Boi+21, Section 3.2]). We recall that BGV [BGV14] uses the relinearization key $\mathsf{rk} = (-(a \cdot s + t \cdot e + s^2, a)$, which is a valid encryption of s^2 under s.

```
$\mathcal{A}_1^{\mathsf{KGen},\mathsf{Dec}}(\mathsf{pk})$

$\mathsf{pk}_{\mathsf{id}} \leftarrow \mathcal{O}_{\mathsf{KGen}}(\mathsf{id})$
$c_x := \mathsf{pk.rk}$
$(\sigma_x, \tau_x) := (c_x, c_x)$
$\sigma_{s_2} = (c_y, \pi) \leftarrow \mathsf{Compute}_{\mathsf{pk}_{\mathsf{id}}}(\sigma_x)$
$s_2 \leftarrow \mathcal{O}_{\mathsf{Dec}}(\sigma_s, \tau_x)$
$s := \mathsf{GetSquareRoot}(s_2)$
$x_0 := 0; x_1 := 1$
**return** $x_0, x_1$
```

```
$\mathcal{A}_2^{\mathsf{KGen}}(\widehat{\sigma}, \widehat{\tau})$

$\widehat{x} := \mathsf{Dec}_s(\widehat{\sigma})$
**return** $\widehat{x} = x_1$
```

$\mathsf{GetSquareRoot}$ can be implemented efficiently by applying the inverse NTT transform, finding the square root modulo q_i of each component (e.g., by using the Tonelli-Shanks algorithm), and then applying the NTT transform to recover s.

One could argue that the VC scheme used in Attacks 1 and 2 was too weak, and that a VC scheme achieving the stronger *context-hiding* property for non-deterministic computations may have thwarted this attack. As exposed in Chapter 3, the context-hiding property gives the two following guarantees:

(a) No information on (x, w) can be inferred from $\tau_x, \mathsf{sk}, \mathsf{sk}_f$, and σ_y.

(b) No information on w can be inferred from $\tau_x, \mathsf{sk}, \mathsf{sk}_f$, σ_y, and σ_x.

Context-hiding thus guarantees that given access to the secret keys and to the output of the computation, an adversary is not able to infer information about the inputs (e.g., by analysing the noise). These guarantees are orthogonal to the attacks outlined above, and do therefore not protect against key-recovery attacks per se. We summarise these three settings in Figure 5.1:

(a) Figure 5.1a shows a setting where Alice sends her encrypted input to the server, which evaluates the circuit and forwards the result to Bob. Even if Bob is malicious, he should not be able to recover anything about x and w that it could not recover by knowing y.

(b) Figure 5.1b shows a typical client-server secure function evaluation scenario, where Alice also takes on the role of decryptor. Here, a malicious Alice should not be able to learn any additional information about the server's input w.

(c) Figure 5.1c shows the same client-server interaction, but here Alice functions as a decryption oracle for the server, and a malicious server should not be able to later distinguish between two encryptions of different plaintexts. In particular, the server should not be able to recover Alice's secret key.

$$\begin{tikzcd}
& S(w) \arrow[dr, "\sigma_y"] & \\
A(x) \arrow[ur, "\sigma_x"] \arrow[rr, "\tau_x"] & & B_{\mathsf{sk},\mathsf{sk}_f}
\end{tikzcd}$$

(a) $B_{\mathsf{sk},\mathsf{sk}_f}(\tau_x, \sigma_y) \not\Rightarrow (x, w)$

$$\begin{tikzcd}
S(w) \arrow[d, bend left, "\sigma_y"] \\
A_{\mathsf{sk},\mathsf{sk}_f}(x) \arrow[u, bend left, "\sigma_x"]
\end{tikzcd}$$

(b) $A_{\mathsf{sk},\mathsf{sk}_f}(\tau_x, \sigma_y, \sigma_x) \not\Rightarrow w$

$$\begin{tikzcd}
S(w) \arrow[d, bend left, "\sigma_y"] \\
A(x) \arrow[u, bend left, "\sigma_x"] \arrow[u, dashed, bend right=60, "y"']
\end{tikzcd}$$

(c) $S^{\mathsf{Dec}}(\sigma_x) \not\Rightarrow \mathsf{sk}$

Figure 5.1: Guarantees given by context-hiding (Figures 5.1a and 5.1b), and by input privacy in the face of decryption oracles (Figure 5.1c). Malicious parties are shown in red.

We have thus seen that existing integrity notions for FHE do not provide the functional guarantees required for vast classes of FHE applications in realistic settings. In addition, existing notions and generic constructions do not even guarantee input privacy against realistic malicious adversaries. In order to remediate this issue, we now formally define new integrity notions that capture the requirements of modern FHE, and provide generic constructions that achieve them in the next sections.

5.3 Integrity Notions for Real-World FHE

The attacks presented in Sections 5.1 and 5.2 are generic and do not rely on any specific concrete constructions; rather, they exploit fundamental shortcomings in the integrity notions that VC schemes in the literature aim to achieve. In order to generically defend against such attacks, we strengthen and extend the unified VC notion presented in Chapter 3 to enforce robustness in the face of malicious adversaries with access to decryption oracles, and to require more comprehensive security guarantees.

5.3.1 Property 1: Maliciously-Secure VC against Decryption Oracles

The first property we introduce is *maliciously-secure Verifiable Computation with decryption oracles*, which captures realistic adversaries for real-world FHE deployments, and is generally desirable regardless of the specific FHE application.

Formally, a VC-CCA1 scheme is a VC scheme that satisfies the following definition of input privacy:

Correctness. Correctness as defined for VC schemes in the literature (Definition 3.2) de facto excludes approximate FHE schemes, which have been used in a considerable number of systems (particularly for machine learning applications).

Definition 5.1 (Correctness for Non-Deterministic and Approximate Computations) *Correctness guarantees that an honest verifier will always accept a result computed by an honest worker. More formally, for all functions f, and for all x, w in the domain of f:*

$$\Pr\left[\begin{array}{c|c} \mathsf{Verify}_{\mathsf{k}}(\tau_x, \sigma_y) = 1 \land & (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\kappa) \\ & (\mathsf{pk}_f, \mathsf{sk}_f) \leftarrow \mathsf{KGen}(f) \\ \left\|\mathsf{Decode}_{\mathsf{sk},\mathsf{sk}_f}(\sigma_y, w) - f(x, w)\right\| \leq \varepsilon & (\sigma_x, \tau_x) \leftarrow \mathsf{ProbGen}_{\mathsf{key}}(x) \\ & \sigma_y \leftarrow \mathsf{Compute}_{\mathsf{pk}_f}(\sigma_x, w) \end{array}\right] = 1,$$

where $\|\cdot\|$ is a scheme-specific norm, and ε is a scheme-specific upper bound on the decoding error (which may depend on f, pk, pk_f, or other quantities of the scheme).

Definition 5.1 naturally generalises deterministic computations ($w = \emptyset$) and exact computations ($\epsilon = 0$).

In addition, some applications also require that their inputs conform to some checks, e.g., that they lie in some numerical range. No VC scheme in the literature supports this functionality. Therefore, we define an even stronger correctness notion, by requiring that for in addition to a function f, correctness requires a predicate $\Phi : (x, w) \mapsto 0/1$, which encodes such checks. The requirement for correctness with input checks thus becomes:

$$\Pr\left[\begin{array}{c|c} \mathsf{Verify}_{\mathsf{k}}(\tau_x, \sigma_y) = 1 \land & (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\kappa) \\ \Phi(w, x) = 1 \land & (\mathsf{pk}_f, \mathsf{sk}_f) \leftarrow \mathsf{KGen}(f) \\ \left\|\mathsf{Decode}_{\mathsf{sk},\mathsf{sk}_f}(\sigma_y, w) - f(x, w)\right\| \leq \varepsilon & (\sigma_x, \tau_x) \leftarrow \mathsf{ProbGen}_{\mathsf{key}}(x) \\ & \sigma_y \leftarrow \mathsf{Compute}_{\mathsf{pk}_f}(\sigma_x, w) \end{array}\right] = 1,$$

While such checks could in principle be incorporated into the function f, this would limit us to predicates expressible as arithmetic circuits. We therefore express them separately, as they might also be implemented using a separate mechanism.

Input Privacy. We now strengthen the definition of input privacy to hold in the face of adversaries with access to a decryption oracle.

Definition 5.2 (PRIV-CCA1)
A scheme $\mathcal{VC}$ is PRIV-CCA1*-secure (i.e., provides* CCA1 *input privacy) if the advantage of any PPT adversary $\mathcal{A}$ (defined as in Definition 3.5 in the game* $\mathsf{Expr}^{\mathsf{Priv}}[\mathcal{A}](1^\kappa)$ *is negligible:*

```
$\mathsf{Expr}_b^{\mathsf{PRIV\text{-}CCA1}}[\mathcal{A}](1^\kappa)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\kappa)$
$(x_0, x_1) \leftarrow \mathcal{A}_1^{\mathsf{KGen},\mathsf{ProbGen},\mathsf{Dec}}(\mathsf{pk}, 1^\kappa)$
$(\sigma_b, \tau_b) \leftarrow \mathsf{ProbGen}_{\mathsf{key}}(x_b)$
$\widehat{b} \leftarrow \mathcal{A}_2^{\mathsf{ProbGen}}(\mathsf{pk}, \sigma_b, \tau_b)$
return $b = \widehat{b}$
```

```
$\mathcal{O}_{\mathsf{Dec}}^{\mathsf{PRIV\text{-}CCA1}}(\tau_x, \sigma_y)$
$b \leftarrow \mathsf{Verify}_{\mathsf{key}}(\tau_x, \sigma_y)$
if $b = 0$ :
  return $\perp$
else :
  $y \leftarrow \mathsf{Decode}_{\mathsf{sk},\mathsf{sk}_f}(\sigma_y)$
  return $y$
```

```
$\mathcal{O}_{\mathsf{KGen}}^{\mathsf{PRIV\text{-}CCA1}}(f)$   // Only queried once
$(\mathsf{pk}_f, \mathsf{sk}_f) \leftarrow \mathsf{KGen}_{\mathsf{pk}}(f)$
return $\mathsf{pk}_f$
```

```
$\mathcal{O}_{\mathsf{ProbGen}}^{\mathsf{PRIV\text{-}CCA1}}(x)$
$(\sigma_x, \tau_x) \leftarrow \mathsf{ProbGen}_{\mathsf{key}}(x)$
return $(\sigma_x, \tau_x)$
```

Security. Similarly, we redefine VC security to hold in the presence of a malicious adversary with access to a decryption oracle:

Definition 5.3 (VER-CCA1)
A VC scheme is VER-CCA1*-secure if the advantage of any PPT adversary $\mathcal{A}$ in the following game* $\mathsf{Expr}^{\mathsf{Ver}}[\mathcal{A}](1^\kappa)$ *is negligible:*

$$\mathsf{Adv}^{\mathsf{Ver}}[\mathcal{A}](\kappa) = \Pr\Big[\mathsf{Expr}^{\mathsf{Ver}}[\mathcal{A}](1^\kappa) = 1\Big] = \mathsf{negl}(\kappa)$$

Here, $\mathcal{O}_{\mathsf{KGen}}$*,* $\mathcal{O}_{\mathsf{ProbGen}}$*, and* $\mathcal{O}_{\mathsf{Dec}}$ *are as in Definition 5.2.*

```
$\mathsf{Expr}^{\mathsf{VER\text{-}CCA1}}[\mathcal{A}](1^\kappa)$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\kappa)$
$x \leftarrow \mathcal{A}_1^{\mathsf{KGen},\mathsf{ProbGen},\mathsf{Dec}}(\mathsf{pk})$
$(\sigma_x, \tau_x) \leftarrow \mathsf{ProbGen}_{\mathsf{key}}(x)$
$(f, \sigma_y) \leftarrow \mathcal{A}_2^{\mathsf{KGen},\mathsf{ProbGen}}(\sigma_x, \tau_x)$
$b \leftarrow \mathsf{Verify}_{\mathsf{k}}(\tau_x, \sigma_y)$
**return** $(b = 1) \wedge (\nexists w : \sigma_y = \mathsf{Compute}_{\mathsf{pk}_f}(\sigma_x, w))$
```

Contrary to Definition 3.4, we modify the winning condition of the adversary to $\sigma_y = \mathsf{Compute}_{\mathsf{pk}_f}(\sigma_x, w))$, which allows us to support approximate computations as well. For exact computation, we recover the previous definition by observing that $\sigma_y = \mathsf{Compute}_{\mathsf{pk}_f}(\sigma_x, w)) \Rightarrow \mathsf{Decode}_{\mathsf{sk},\mathsf{sk}_f}(\sigma_y) = f(x, w)$.

Context-Hiding. A vast majority of FHE use cases require inputs from multiple parties, and it is thus paramount that a VC scheme that seeks to be general offers support for private inputs of multiple parties. This excludes most VC schemes in the literature, and leaves only the approach of Bois et al. [Boi+21].

The definition of context-hiding for non-deterministic computation introduced in [Boi+21] (see Definition 3.6) is tailored for VC schemes with public verifiability. In particular, property (a) in Definition 3.6 only makes sense if the scheme is publicly verifiable, and if any party other than the encryptor is involved in the protocol. This is not the case for many client-server FHE applications (e.g., PIR, PSI/PSU, ML inference), and we can thus define a weaker notion of context-hiding that captures this setting by only considering the case (b) of Definition 3.6:

Definition 5.4 (server context-hiding [FNP20; Boi+21])
Formally, server context-hiding is defined using the same simulators as in Definition 3.6, except for S_τ simulators. A $\mathcal{VC}$ scheme is server context-hiding if there exist efficient simulator algorithms S_1, S_2 and S_3 as in Definition 3.6.

While very useful in deterministic computations, computation integrity gives little guarantees for computations with inputs from multiple parties. Indeed, for a client with input x in a multi-party application, existing VC schemes only guarantees that there exists an input w such that the result y satisfies $y = f(x, w)$, but gives no guarantees about w. As we explored in Section 5.1, many FHE use cases in practice require additional guarantees, which we outline in the following.

5.3.2 Property 2: Maliciously-Secure VC with Input Commitment

For applications with repeated queries and with the requirement that the same circuit and the same inputs from other parties be applied each time, input commitment is desirable. To the best of our knowledge, input commitment is not supported by most of the existing VC schemes; in [FNP20] Fiore et al. use a Commit-and-Prove SNARK internally to build their VC, but no mention of using it to commit to some inputs before a protocol run is made.

Formally, we define this input commitment as a forgery game, i.e., it is unfeasible for an adversary to come up with a function f, an input x, and two different outputs $w \neq w'$ such that the output of the computation with w and the computation with w' both pass the verification check:

Definition 5.5 (with Input Commitment)
Let $\mathcal{VC}$ be a Verifiable Computation scheme. $\mathcal{VC}$ provides input commitment *if, for any PPT adversary $\mathcal{A}$, the probability that the following experiment returns 1 is negligible.*

```
$\mathsf{Expr}^{\mathsf{COM}}[\mathcal{A}](1^\kappa)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\kappa)$
$(x, w, w') \leftarrow \mathcal{A}^{\mathsf{KGen},\mathsf{ProbGen},\mathsf{Dec}}(1^\kappa, \mathsf{pk})$
$(\sigma_x, \tau_x) \leftarrow \mathsf{ProbGen}_{\mathsf{key}}(x)$
$\sigma_y \leftarrow \mathsf{Compute}_{\mathsf{pk}_f}(\sigma_x, w)$
$\sigma'_y \leftarrow \mathsf{Compute}_{\mathsf{pk}_f}(\sigma_x, w')$
$b \leftarrow \mathsf{Verify}_{\mathsf{key}}(\tau_x, \sigma_y)$
$b' \leftarrow \mathsf{Verify}_{\mathsf{key}}(\tau_x, \sigma'_y)$
$\mathbf{return}\ (w \neq w') \wedge (b = 1) \wedge (b' = 1)$
```

The KGen, ProbGen, and Dec oracles in Definition 5.5 are the same as in Definitions 5.2 and 5.3.

5.3.3 Property 3: Maliciously-Secure VC with Proof-of-Computation

VC does not per se give a proof that any computation has taken place, only that the result returned to the client is equal to some (potentially non-deterministic) function evaluated on the client's input. We formalise Proof-of-Computation as follows:

Every circuit f can be trivially extended to a bigger and deeper circuit $\tilde{f}$ taking more server inputs (e.g., by adding multiplication gates by an all-1 value). However, existing VC schemes only guarantee circuit-satisfiability, i.e., that the output of each gate corresponds to the gate's operation applied to the gate inputs. This does however not imply that the server actually performed any computation, as the server could have computed f honestly, and then simply set the output of the remaining gates of $\tilde{f}$ to make $\tilde{f}$ satisfiable, without computing any of the new intermediate values. A malicious server could therefore use this trick in order to bill the client more without expending any more compute (e.g., by claiming to have evaluated a much deeper machine learning model). To remedy this, the client would want a guarantee that the server expended a similar amount of computation as what is needed to compute the claimed circuit $\tilde{f}$. No known VC scheme for FHE support such guarantees.

Definition 5.6 (Proof-of-Computation)
Formally, a VC scheme offers Proof-of-Computation if for any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ with $\mathcal{A}_2$ with concrete runtime $\widehat{T}$ (expressed in terms of elementary operations), the probability that the following experiment returns 1 is negligible.

```
$\mathsf{Expr}^{\mathsf{PoC}}[\mathcal{A}](1^\kappa)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\kappa)$
$x \leftarrow \mathcal{A}_1^{\mathsf{ProbGen},\mathsf{Dec}}(\mathsf{pk})$
$(\sigma_x, \tau_x) \leftarrow \mathsf{ProbGen}_{\mathsf{key}}(x)$
$(f, w, \sigma_y) \leftarrow \mathcal{A}_2^{\mathsf{ProbGen},\mathsf{Dec}}(\sigma_x, \tau_x)$
$b \leftarrow \mathsf{Verify}_{\mathsf{key}}(\tau_x, \sigma_y)$
$T := T[\mathsf{Compute}_{\mathsf{pk}_f}(\sigma_x, w)]$
$\textbf{return } (b = 1) \wedge (T < \widehat{T})$
```

T here is the concrete runtime of honestly evaluating $\mathsf{Compute}_{\mathsf{pk}_f}(\sigma_x, w)$ *(under the same computation model as $\mathcal{A}$).*

Having formally introduced these stronger and more expressive VC notions, we now investigate more traditional indistinguishability-based security notions for FHE in Chapter 6, and show how they are connected to VC. After that, we will show generic constructions to achieve our newly introduced properties for state-of-the-art schemes in Chapter 7, and sketch efficient and expressive integrity primitives for those generic constructions in Chapter 8.

Chapter 6

Tidying up the FHE Security Zoo

After defining more robust VC notions for real-world FHE in chapter 5, we will now investigate more classical indistinguishability-based security notions for FHE (used extensively for public-key cryptography, and since adapted for FHE schemes). In section 6.1, we survey the literature for FHE constructions and attacks, and compile a list of existing security notions for FHE used in existing works. Then, in section 6.2, we then uncover several definitional issues and ambiguities in the $\mathsf{IND\text{-}CPA^D}$ notion, and refine it. Finally, in sections 6.3 and 6.4, we relate VC security notions and the more traditional $\mathsf{IND}\text{-}*$ security notions.

6.1 The FHE Security Zoo

Over the years, a plethora of security notions for FHE have been proposed, both adaptations of established notions initially introduced for (non-homomorphic) public-key encryption, as well as new notions tailored to certain families of FHE schemes.

6.1.1 $\mathsf{IND}\text{-}\{\mathsf{CPA}, \mathsf{CVA1}, \mathsf{CVA2}, \mathsf{CCA1}, \mathsf{CCA1.5}, \mathsf{CCA2}\}$

We introduce the following general experiment $\mathsf{Expr}_b^{\mathsf{IND\text{-}ATK}}[\mathcal{A}](1^\kappa)(1^\kappa)$ for the security notion $\mathsf{IND\text{-}ATK}$, parametrised by the set $\mathcal{O}_1$ (respectively $\mathcal{O}_2$) of oracles provided to the adversary $\mathcal{A}$ before it receives the challenge ciphertext $\mathbf{ct}^*$ (respectively after). Greyed out code prevents trivial attacks.

$$\begin{tikzcd}
 & & \mathsf{IND\text{-}CCA2} \arrow[d, shift right] & & \\
 & & \mathsf{IND\text{-}CCA1.5} \arrow[u, shift right, "/" marking] \arrow[dl] \arrow[dr] & & \\
 & \mathsf{IND\text{-}CVA2} \arrow[ur, shift right, "/" marking] \arrow[dr] & & \mathsf{IND\text{-}CCA1} \arrow[ul, shift right, dashed, "/" marking] \arrow[dl] \arrow[dr] & \\
 & & \mathsf{IND\text{-}CVA1} \arrow[ul, shift right, dashed, "/" marking] \arrow[ur, shift right, "/" marking] \arrow[dr] & & \mathcal{F}\text{-}\mathsf{IND\text{-}CPA^D} \arrow[ul, shift right, "/" marking] \arrow[dl] \\
 & & & \mathsf{IND\text{-}CPA} \arrow[ul, shift right, dashed, "/" marking] \arrow[ur, shift right, dotted, "/" marking] &
\end{tikzcd}$$

Figure 6.1: The FHE security zoo. Arrows denote implications, striked-through arrows denote separations.
Dashed arrows denote separations which do not hold for unprotected FHE schemes, and dotted arrows those that do not hold for exact FHE schemes.

```
$\mathsf{Expr}_b^{\mathsf{IND\text{-}ATK}}[\mathcal{A}](1^\kappa)$

$(\mathsf{pk}, \mathsf{ek}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\kappa)$
$(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{A}_1^{\mathcal{O}_1}(1^\kappa, \mathsf{pk}, \mathsf{ek})$   // Phase 1
$\mathbf{ct}^* \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{m}_b)$
$\widehat{b} \leftarrow \mathcal{A}_2^{\mathcal{O}_2}(\mathbf{ct}^*)$   // Phase 2
```

```
$\mathcal{O}_{\mathsf{Verify}}^{\mathsf{IND\text{-}ATK}}(\mathbf{ct})$

if $\mathbf{ct} = \mathbf{ct}^*$ :
    return forbidden
$\mathbf{m} \leftarrow \mathsf{Dec}_{\mathsf{sk}}(\mathbf{ct})$
if $\mathbf{m} = \perp$ :
return $[m = \perp]$
```

```
$\mathcal{O}_{\mathsf{Dec}}^{\mathsf{IND\text{-}ATK}}(\mathbf{ct})$

if $\mathbf{ct} = \mathbf{ct}^*$ :
    return forbidden
$\mathbf{m} \leftarrow \mathsf{Dec}_{\mathsf{sk}}(\mathbf{ct})$
return $\mathbf{m}$
```

For every $\mathsf{IND\text{-}ATK}$ experiment, we define a corresponding attack game in which the challenger randomly samples a bit b and runs $\mathsf{Expr}_b^{\mathsf{IND\text{-}ATK}}[\mathcal{A}](1^\kappa)$ before outputting the adversary's guess $\widehat{b}$. We define the adversary's advantage in this attack game as $\mathsf{Adv}^{\mathsf{IND\text{-}ATK}}[\mathcal{A}](\kappa) = 2\left|\Pr\left[b = \widehat{b}\right] - \frac{1}{2}\right|$.

Table 6.1 summarises the different instantiations of $\mathsf{IND\text{-}ATK}$ in the literature, through which we will now go into more detail below.

Indistinguishability under Chosen Plaintext Attack ($\mathsf{IND\text{-}CPA}$) is the weakest notion, and is equivalent to semantic security. The goal of the adversary $\mathcal{A}$ is to guess if given ciphertext $\mathbf{ct}$ encrypts the plaintext $\mathbf{m}_0$ or $\mathbf{m}_1$, where both plaintexts are chosen by $\mathcal{A}$. All known FHE schemes satisfy this notion (and all state-of-the-art schemes satisfy only this notion).

Full name	ATK	$\mathcal{O}_1$	$\mathcal{O}_2$
Chosen Plaintext Attack	CPA		
non-adaptive Chosen Verification Attack	CVA1	$\mathcal{O}_{\mathsf{Verify}}$	
non-adaptive Chosen Ciphertext Attack	CCA1	$\mathcal{O}_{\mathsf{Dec}}$	
adaptive Chosen Verification Attack	CVA2	$\mathcal{O}_{\mathsf{Verify}}$	$\mathcal{O}_{\mathsf{Verify}}$
adaptive Chosen Decryption/Verification Attack	CCA1.5	$\mathcal{O}_{\mathsf{Dec}}$	$\mathcal{O}_{\mathsf{Verify}}$
adaptive Chosen Ciphertext Attack	CCA2	$\mathcal{O}_{\mathsf{Dec}}$	$\mathcal{O}_{\mathsf{Dec}}$

Table 6.1: Oracles available to an adversary for different instantiations of the IND-ATK notion.

Chosen Verification Attacks (IND-CVA1 and IND-CVA2) games grant the adversary access to a *verification oracle* $\mathcal{O}_{\mathsf{Verify}}(\mathbf{ct})$, which returns 1 if $\mathsf{Dec}_{\mathsf{sk}}(\mathbf{ct}) = \perp$, and 0 otherwise. For IND-CVA1 the adversary only gets access to this oracle before receiving the challenge ciphertext, while the oracle is accessible at all times for IND-CVA2. For all known FHE schemes, Dec will not return $\perp$ for any element in the ciphertext space $\mathcal{C}$, as no checks are performed. The verification oracle is therefore meaningless for unprotected FHE schemes, and is only useful for more complex constructions with built-in ciphertext validity checks. For these schemes, IND-CPA, IND-CVA1, and IND-CVA2 are equivalent.

Chosen Ciphertext Attacks (IND-CCA1 and IND-CCA2) games model an adversary with access to a *decryption oracle* $\mathcal{O}_{\mathsf{Dec}}(\mathbf{ct})$, which return $\mathsf{Dec}_{\mathsf{sk}}(\mathbf{ct})$. These notions are therefore stronger than their corresponding CVA equivalents. IND-CCA2 security is fundamentally at odds with the homomorphic property, and no HE scheme can achieve IND-CCA2 security, due to the inherent malleability of the scheme. For any scheme $\mathcal{E}$ that is homomorphic with respect to the operation $\oplus$, an adversary can ask the challenger for an encryption $\mathbf{ct}_0$ of the neutral plaintext element $\mathbf{0} \in \mathcal{M}$ and submit $\mathbf{ct}' = \mathbf{ct}^* \oplus \mathbf{ct}_0$ to the decryption oracle. Since $\mathbf{ct}' \neq \mathbf{ct}^*$ the decryption oracle will return $\mathsf{Dec}_{\mathsf{sk}}(\mathbf{ct}^* \oplus \mathbf{ct}_0) = \mathbf{m}^* \oplus \mathbf{0} = \mathbf{m}^*$ and can trivially recover b by comparing $\mathbf{m}^*$ to its chosen plaintexts $\mathbf{m}_0$ and $\mathbf{m}_1$. Interestingly, Das et al. showed in [DDA13] that IND-CCA1 and IND-CVA2 are incomparable, i.e., there exist schemes that are IND-CCA1-secure but not IND-CVA2-secure, and vice-versa.

Indistinguishability under adaptive Chosen Ciphertext Decryption/Verification Attack (IND-CCA1.5) (introduced in [DDA13]) is a stronger notion than both IND-CVA2 and IND-CCA1, where the adversary is given access to a full decryption oracle in the first phase, and to a verification oracle in the second phase. An interesting result by Das et al. [DDA13] is that IND-CCA1.5 is not implied by IND-CCA1 and IND-CVA1 together, i.e., there exist schemes that are IND-CCA1 and IND-CVA2 secure, but that are not IND-CCA1.5 secure.

In the context of FHE integrity, we are particularly interested in the CCA1 setting, as IND-CCA1-security directly implies resistance against key-recovery attacks. While an adversary with access to a full decryption oracle will inevitably be able to decrypt ciphertexts and learn the corresponding plaintexts, the oracle in practice may only partial or hard to trigger. Therefore, for well-defended (but unprotected) deployments it might not be feasible in practice for an adversary to get a full decryption for each plaintext it wants to learn. However, if a key-recovery attack is possible, the adversary may be able to recover the secret key (potentially over many queries), and gain the ability to decrypt all ciphertexts after recovery.

6.1.2 IND-CPA$^{\mathsf{D}}$

In [LM21], Li and Micciancio showed that some approximate FHE schemes (in particular, CKKS) are vulnerable to key recovery attacks in the presence of a (weak) decryption oracle (while still achieving IND-CPA security).

To address this gap between theory and practice, the authors introduce the IND-CPA$^{\mathsf{D}}$ notion (defined below), which allows the adversary to query a decryption oracle for honestly-generated ciphertexts (i.e., generated by a valid encryption, or by evaluating a function on valid ciphertexts).

Definition 6.1 (IND-CPA$^{\mathsf{D}}$, rephrased from [LM21; Li+22])
An FHE scheme is IND-CPA$^{\mathsf{D}}$*-secure if every adversary* $\mathcal{A}$ *has negligible advantage* $\mathsf{Adv}^{\mathsf{IND\text{-}CPA^D}}[\mathcal{A}](\kappa) := 2\left|\Pr\left[b = \widehat{b}\right] - \frac{1}{2}\right|$ *in the attack game with the following experiments and oracles:*

```
$\mathsf{Expr}_b^{\mathsf{IND\text{-}CPA^D}}[\mathcal{A}](1^\kappa)(1^\kappa)$
    $(\mathsf{pk}, \mathsf{ek}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\kappa)$
    $S := [\ ]$
    $i := 0$
    $\widehat{b} \leftarrow \mathcal{A}^{\mathsf{Enc},\mathsf{Eval},\mathsf{Dec}}(1^\kappa, \mathsf{pk}, \mathsf{ek})$

$\mathcal{O}_{\mathsf{Enc}}^{\mathsf{IND\text{-}CPA^D}}[\mathsf{pk}](\mathbf{m}_0, \mathbf{m}_1)$
    $\mathbf{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{m}_b)$
    $S[i] := (\mathbf{m}_0, \mathbf{m}_1, \mathbf{ct})$
    $i := i + 1$
    return $\mathbf{ct}$

$\mathcal{O}_{\mathsf{Eval}}^{\mathsf{IND\text{-}CPA^D}}[\mathsf{ek}](f, i_1, \ldots, i_k)$
    if $f \notin \mathcal{F}$ : return $\bot$
    $\mathbf{ct} \leftarrow \mathsf{Eval}_{\mathsf{ek}}(f, S[i_1].\mathbf{ct}, \ldots, S[i_k].\mathbf{ct})$
    $\mathbf{r}_0 := f(S[i_1].\mathbf{m}_0, \ldots, S[i_k].\mathbf{m}_0)$
    $\mathbf{r}_1 := f(S[i_1].\mathbf{m}_1, \ldots, S[i_k].\mathbf{m}_1)$
    $S[i] := (\mathbf{r}_0, \mathbf{r}_1, \mathbf{ct})$
    $i := i + 1$
    return $\mathbf{ct}$

$\mathcal{O}_{\mathsf{Dec}}^{\mathsf{IND\text{-}CPA^D}}[\mathsf{sk}](j)$
    if $S[j].\mathbf{m}_0 = S[j].\mathbf{m}_1$ :
        $\mathbf{m} \leftarrow \mathsf{Dec}_{\mathsf{sk}}(S[j].\mathbf{ct})$
        return $\mathbf{m}$
    else :
        return $\bot$
```

Here, $\mathcal{F}$ *is the set of all functions or circuits supported by the scheme, and* S *and* i *are part of the challenger's state which the adversary can read from,* *but not write to. As in [LM21], the evaluation and decryption oracles return $\perp$ if queried with out-of-bounds indices. However, we make the check $f \in \mathcal{F}$ explicit, whereas it was implicit in [LM21].*

Li and Micciancio prove that $\mathsf{IND\text{-}CPA^D}$ reduces to $\mathsf{IND\text{-}CPA}$ for exact schemes, but that this equivalence does not hold for approximate schemes. In particular, CKKS is $\mathsf{IND\text{-}CPA}$-secure, but not $\mathsf{IND\text{-}CPA^D}$-secure. In follow-up work, Li and Micciancio introduce a generic paradigm to transform any approximate $\mathsf{IND\text{-}CPA}$-secure scheme with well-behaved decryption errors into an $\mathsf{IND\text{-}CPA^D}$-secure FHE scheme by using differential privacy.

Unfortunately, the $\mathsf{IND\text{-}CPA^D}$ definition as introduced in [LM21] suffers from several ambiguities which makes it difficult to work with in practice and does not fully capture the behaviour of state-of-the-art FHE schemes (in particular "exact" FHE schemes). In the next section, we detail these definitional issues, and we introduce an alternative definition of $\mathsf{IND\text{-}CPA^D}$ that tries to capture the same properties, but conforms better with existing FHE schemes.

6.2 $\mathcal{F}$-$\mathsf{IND\text{-}CPA^D}$: Refining $\mathsf{IND\text{-}CPA^D}$

While it captures a new and important intermediate security notion between $\mathsf{IND\text{-}CPA}$ and $\mathsf{IND\text{-}CCA1}$, the $\mathsf{IND\text{-}CPA^D}$ security notion introduced by Li and Micciancio suffers from several definitional issues, which make it ill-suited for use in security proofs.

First, without bootstrapping, none of the known FHE schemes are unconditionally exact; for a given set of parameters, an FHE scheme is exact only for a specific set of circuits $\mathcal{F}$. Additionally, this set of circuit for which the scheme is exact is not closed under composition; repeatedly application the same elementary operation (say, and addition) to freshly encrypted inputs will lead to correct decryptions up to a certain point (as long as the noise accumulated in the resulting ciphertext is below some threshold), after which all decryptions will be erroneous. This clashes with the experiment as defined in [LM21], where the adversary is allowed to evaluate any circuit in $\mathcal{F}$ repeatedly on any of the ciphertexts in the challenger's S. Finally, in [LM21] functions are defined over plaintexts, and a correspondence with an equivalent function over ciphertexts is assumed implicitly. With FHE however, a circuit over plaintexts can have multiple equivalent circuits over ciphertexts (notably depending on if/where ciphertext maintenance operations are inserted). If a scheme supports bootstrapping, there are even infinitely many ciphertexts (each with a different number of bootstrapping operations performed on the result) corresponding to the same plaintext circuit! As ciphertext maintenance operations can drastically influence the noise present in ciphertexts (and thus whether the ciphertext decrypts correctly or not), this ambiguity makes the $\mathsf{IND\text{-}CPA^D}$ notion as defined in [LM21] unwieldy, and calls for a more precise definition, which we introduce in the next subsection.

Definition 6.2 ($\mathcal{F}$-perfect-correctness)
A scheme $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ is $\mathcal{F}$-perfectly-correct for a family of functions $\mathcal{F}$ if for every $F : \mathcal{C}^k \to \mathcal{C}$ in $\mathcal{F}$ and for every $\mathbf{m}_1, \ldots, \mathbf{m}_m, \mathbf{m}'_1, \ldots, \mathbf{m}'_n$ in $\mathcal{M}$, it holds that

$$f(M_{in}, M'_{in}) = \mathsf{Dec}_{\mathsf{sk}}\left(\mathsf{Eval}_{\mathsf{ek}}\left(F, C_{in}, M'_{in}\right)\right)$$

with probability 1, where $C_{in} = (\mathsf{Enc}_{\mathsf{pk}}(\mathbf{m}_1), \ldots, \mathsf{Enc}_{\mathsf{pk}}(\mathbf{m}_m))$, $M_{in} = (\mathbf{m}'_1, \ldots \mathbf{m}'_n)$, $(\mathsf{pk}, \mathsf{ek}, \mathsf{sk} \leftarrow \mathsf{KGen}(1^\kappa)$, and where f is the (unique) plaintext circuit equivalent to the ciphertext circuit F.

Some FHE schemes are not $\mathcal{F}$-perfectly-correct, but are instead *approximate*, meaning that the decrypted result is "close" to the plaintext result:

Definition 6.3 ($\mathcal{F}$-approximativity)
A scheme $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ is $\mathcal{F}$-perfectly-correct for a family of functions $\mathcal{F}$ if for every $F : \mathcal{C}^k \to \mathcal{C}$ in $\mathcal{F}$ and for every $\mathbf{m}_1, \ldots, \mathbf{m}_m, \mathbf{m}'_1, \ldots, \mathbf{m}'_n$ in $\mathcal{M}$, it holds that

$$\|f(M_{in}, M'_{in}) - \mathsf{Dec}_{\mathsf{sk}}\left(\mathsf{Eval}_{\mathsf{ek}}\left(F, C_{in}, M'_{in}\right)\right)\| \le \varepsilon(F, C_{in}, M'_{in})$$

where $C_{in} = (\mathsf{Enc}_{\mathsf{pk}}(\mathbf{m}_1), \ldots, \mathsf{Enc}_{\mathsf{pk}}(\mathbf{m}_m))$, $M_{in} = (\mathbf{m}'_1, \ldots \mathbf{m}'_n)$, $(\mathsf{pk}, \mathsf{ek}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\kappa)$, and where f is the (unique) plaintext circuit equivalent to the ciphertext circuit F.

Here, ε is an error function depending on the circuit $FitsinputsC_{in}andM_{in}$' and the public key (sometimes also on the secret key)

Definition 6.4 ($\mathcal{F}$-$\mathsf{IND\text{-}CPA^D}$)
An FHE scheme is $\mathcal{F}$-$\mathsf{IND\text{-}CPA^D}$-secure for a set of circuits $\mathcal{F}$ if every adversary $\mathcal{A}$ has negligible advantage $\mathsf{Adv}^{\mathcal{F}\text{-}\mathsf{IND\text{-}CPA^D}}[\mathcal{A}](\kappa) := 2\left|\Pr\left[b = \widehat{b}\right] - \frac{1}{2}\right|$ in the attack game with the following experiments and oracles:

```
$\mathsf{Expr}_b^{\mathcal{F}\text{-}\mathsf{IND}\text{-}\mathsf{CPA}^{\mathsf{D}}}[\mathcal{A}](1^\kappa)$
$(\mathsf{pk}, \mathsf{ek}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\kappa)$
$S_{in}, S_{out} := [\ ]$
$i_{in}, i_{out} := 0$
$(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{A}_1^{\mathsf{Enc},\mathsf{Eval},\mathsf{Dec}}(1^\kappa, \mathsf{pk}, \mathsf{ek})$
$\mathbf{ct}^* \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{m}_b)$
$S_{in}[i_{in}] := (\mathbf{m}_0, \mathbf{m}_1, \mathbf{ct}^*)$
$i_{in} := i_{in} + 1$
$\widehat{b} \leftarrow \mathcal{A}_2^{\mathcal{O}}(\mathbf{ct}^*)$
return $\left[b = \widehat{b}\right]$
```

```
$\mathcal{O}_{\mathsf{Enc}}^{\mathcal{F}\text{-}\mathsf{IND}\text{-}\mathsf{CPA}^{\mathsf{D}}}[\mathsf{pk}](\mathbf{m}, r)$
$\mathbf{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{m}; r)$
$S_{in}[i_{in}] := (\mathbf{m}, \mathbf{m}, \mathbf{ct})$
$i_{in} := i_{in} + 1$
return ct
```

```
$\mathcal{O}_{\mathsf{Eval}}^{\mathcal{F}\text{-}\mathsf{IND}\text{-}\mathsf{CPA}^{\mathsf{D}}}[\mathsf{ek}](F, i_1, \ldots, i_k)$
if $F \notin \mathcal{F}$ : return $\perp$
$\mathbf{ct} \leftarrow \mathsf{Eval}_{\mathsf{ek}}(F, S_{in}[i_1].\mathbf{ct}, \ldots, S_{in}[i_k].\mathbf{ct})$
$\mathbf{r}_0 := \mathsf{Eval}(F, S_{in}[i_1].\mathbf{m}_0, \ldots, S_{in}[i_k].\mathbf{m}_0)$
$\mathbf{r}_1 := \mathsf{Eval}(F, S_{in}[i_1].\mathbf{m}_1, \ldots, S_{in}[i_k].\mathbf{m}_1)$
$S_{out}[i_{out}] := (\mathbf{r}_0, \mathbf{r}_1, \mathbf{ct})$
$i_{out} := i_{out} + 1$
return ct
```

```
$\mathcal{O}_{\mathsf{Dec}}^{\mathcal{F}\text{-}\mathsf{IND}\text{-}\mathsf{CPA}^{\mathsf{D}}}[\mathsf{sk}](j)$
if $S_{out}[j].\mathbf{m}_0 = S_{out}[j].\mathbf{m}_1$ :
    $\mathbf{m} \leftarrow \mathsf{Dec}_{\mathsf{sk}}(S_{out}[j])$
    return m
else :
    return $\perp$
```

We also define a non-adaptive version of this notion by not providing any oracle to the adversary in the post-challenge phase ($\mathcal{O} = \emptyset$), and an adaptive version, where the adversary has access to all oracles even after the challenge ($\mathcal{O} = \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec}$). The definition of $\mathsf{IND}\text{-}\mathsf{CPA}^{\mathsf{D}}$ in [LM21] is adaptive. We also define variants where the adversary is only allowed to make q queries, for a pre-defined q.

6.2.1 Positioning $\mathcal{F}$-IND-CPA$^{\mathsf{D}}$ in the Security Zoo

We will now show that $\mathcal{F}\text{-}\mathsf{IND}\text{-}\mathsf{CPA}^{\mathsf{D}}$ is a drop-in replacement for the $\mathsf{IND}\text{-}\mathsf{CPA}^{\mathsf{D}}$ notion in the FHE security zoo, by proving that the same implications and separations hold for $\mathcal{F}\text{-}\mathsf{IND}\text{-}\mathsf{CPA}^{\mathsf{D}}$.

$\mathcal{F}\text{-}\mathsf{IND}\text{-}\mathsf{CPA}^{\mathsf{D}} \rightarrow \mathsf{IND}\text{-}\mathsf{CPA}$. Any IND-CPA-adversary against $\mathcal{E}$ is automatically an $\mathcal{F}\text{-}\mathsf{IND}\text{-}\mathsf{CPA}^{\mathsf{D}}$-adversary, and $\mathsf{Adv}^{\mathsf{IND}\text{-}\mathsf{CPA}}[\mathcal{A}](\kappa) = \mathsf{Adv}^{\mathcal{F}\text{-}\mathsf{IND}\text{-}\mathsf{CPA}^{\mathsf{D}}}[\mathcal{A}](\kappa)$.

$\mathsf{IND}\text{-}\mathsf{CPA} \not\rightarrow \mathcal{F}\text{-}\mathsf{IND}\text{-}\mathsf{CPA}^{\mathsf{D}}$. Let $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be the (unpatched) CKKS scheme, which is IND-CPA-secure. We show that Π is not $\mathcal{F}\text{-}\mathsf{IND}\text{-}\mathsf{CPA}^{\mathsf{D}}$-secure.

$\mathcal{A}$ queries its encryption oracle with $\mathbf{m} = 0$ to get $\mathbf{ct} = (a, b) = (a, a \cdot \mathsf{sk} + e)$, and then queries its evaluation oracle with $f = \mathsf{id}$ and $i_1 = 0$ to get $\mathbf{ct}' = \mathbf{ct}$. $\mathcal{A}$ then queries decryption oracle to get $\mathbf{m}' = -a \cdot \mathsf{sk} + b = e$. By computing $\delta = b - e = a \cdot \mathsf{sk}$, $\mathcal{A}$ gets a linear system for sk. By repeating this process n times, $\mathcal{A}$ can recover sk efficiently.

$\mathsf{IND\text{-}CCA1} \to \mathcal{F}\text{-}\mathsf{IND\text{-}CPA}^{\mathsf{D}}$. Any $\mathcal{F}\text{-}\mathsf{IND\text{-}CPA}^{\mathsf{D}}$-adversary against $\mathcal{E}$ is automatically an $\mathsf{IND\text{-}CCA1}$-adversary, and $\mathsf{Adv}^{\mathcal{F}\text{-}\mathsf{IND\text{-}CPA}^{\mathsf{D}}}[\mathcal{A}](\kappa) = \mathsf{Adv}^{\mathsf{IND\text{-}CCA1}}[\mathcal{A}](\kappa)$.

$\mathcal{F}\text{-}\mathsf{IND\text{-}CPA}^{\mathsf{D}} \not\to \mathsf{IND\text{-}CCA1}$. Let $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be an $\mathcal{F}\text{-}\mathsf{IND\text{-}CPA}^{\mathsf{D}}$-secure FHE scheme We construct a scheme $\mathcal{E}'$ such that $\mathcal{E}'$ is $\mathcal{F}\text{-}\mathsf{IND\text{-}CPA}^{\mathsf{D}}$-secure, but not $\mathsf{IND\text{-}CCA1}$-secure as follows, where $\|$ denotes concatenation. $\mathcal{E}'$ is defined as follows:

$\mathcal{E}'.\mathsf{KGen}(1^\kappa)$: uses the same KGen algorithm as $\mathcal{E}$;

$\mathcal{E}'.\mathsf{Enc}_{\mathsf{pk}}(\mathbf{m})$: returns $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{m})\|0$;

$\mathcal{E}'.\mathsf{Eval}_{\mathsf{ek}}(\mathbf{ct})$: parses $\mathbf{ct}'$ as $\mathbf{ct}\|0$ and returns $\mathcal{E}.\mathsf{Eval}_{\mathsf{ek}}(\mathbf{ct})\|0$;

$\mathcal{E}'.\mathsf{Dec}_{\mathsf{sk}}(\mathbf{ct}')$: returns $\mathsf{Dec}_{\mathsf{sk}}(\mathbf{ct})$ if $\mathbf{ct}' = \mathbf{ct}\|0$, and sk otherwise.

$\mathcal{E}$' is $\mathcal{F}\text{-}\mathsf{IND\text{-}CPA}^{\mathsf{D}}$-secure, as any adversary $\mathcal{A}$ against $\mathcal{E}$' can be immediately translated into an $\mathcal{F}\text{-}\mathsf{IND\text{-}CPA}^{\mathsf{D}}$ adversary $\mathcal{B}$ against $\mathcal{E}$ (Π' is $\mathcal{F}\text{-}\mathsf{IND\text{-}CPA}^{\mathsf{D}}$-secure (by reduction to the $\mathcal{F}\text{-}\mathsf{IND\text{-}CPA}^{\mathsf{D}}$-security of Π)

Π' is not $\mathsf{IND\text{-}CCA1}$-secure : $\mathcal{A}$ queries encryption oracle with $\mathbf{m} = 0$ to get $\mathbf{ct}' = \mathbf{ct}\|0$ and recovers sk by asking for the decryption of $\mathbf{ct}'' = \mathbf{ct}\|1$

6.2.2 Realizing $\mathcal{F}\text{-}\mathsf{IND\text{-}CPA}^{\mathsf{D}}$

We will now show that $\mathcal{F}\text{-}\mathsf{IND\text{-}CPA}^{\mathsf{D}}$ is achievable in practice by adapting the constructions and proofs in [LM21; Li+22] to the $\mathcal{F}\text{-}\mathsf{IND\text{-}CPA}^{\mathsf{D}}$ notion.

$\mathcal{F}\text{-}\mathsf{IND\text{-}CPA}^{\mathsf{D}} = \mathsf{IND\text{-}CPA}$ for $\mathcal{F}$-exact schemes

Theorem 6.5
If a scheme $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ is $\mathsf{IND\text{-}CPA}$-secure, $\mathcal{F}$-exact, and if set membership in $\mathcal{F}$ is efficiently computable, then $\mathcal{E}$ is $\mathcal{F}\text{-}\mathsf{IND\text{-}CPA}^{\mathsf{D}}$-secure.

In particular, for every $\mathcal{F}\text{-}\mathsf{IND\text{-}CPA}^{\mathsf{D}}$-adversary $\mathcal{A}$ for $\mathcal{E}$, there exists an $\mathsf{IND\text{-}CPA}$-adversary $\mathcal{B}$ for $\mathcal{E}$ such that $\mathsf{Adv}^{\mathcal{F}\text{-}\mathsf{IND\text{-}CPA}^{\mathsf{D}}}[\mathcal{A}](\kappa) = \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}[\mathcal{B}](\kappa)$.

Proof (Following [LM21, Lemma 1]) Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a PPT $\mathcal{F}\text{-}\mathsf{IND\text{-}CPA}^{\mathsf{D}}$-adversary attacking $\mathcal{E}$. We construct a PPT $\mathsf{IND\text{-}CPA}$-adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ against $\mathcal{E}$ as follows:

```
$\mathcal{B}_1^{\mathsf{IND\text{-}CPA}}[\mathcal{A}](1^\kappa, \mathsf{pk}, \mathsf{ek})$
    $S_{\text{in}} := [\ ]$
    $i_{\text{in}} := 0$
    $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{A}_1^{\mathsf{Enc},\ \mathsf{Eval},\ \mathsf{Dec}}(1^\kappa, \mathsf{pk}, \mathsf{ek})$
    return $(\mathbf{m}_0, \mathbf{m}_1)$

$\mathcal{B}_2^{\mathsf{IND\text{-}CPA}}[\mathcal{A}](\mathbf{ct}^*)$
    $\widehat{b} \leftarrow \mathcal{A}_2(\mathbf{ct}^*)$
    return $\widehat{b}$
```

```
$\mathsf{EvalSim}[\mathsf{ek}](F, i_1, \ldots, i_k)$
    if $F \notin \mathcal{F}$ : return $\perp$
    $\mathbf{m} \leftarrow \mathsf{Eval}(F, S_{\text{in}}[i_1].\mathbf{m}, \ldots, S_{\text{in}}[i_k].\mathbf{m})$
    $\mathbf{ct} \leftarrow \mathsf{Eval}_{\mathsf{ek}}(F, S_{\text{in}}[i_1].\mathbf{ct}, \ldots, S_{\text{in}}[i_k].\mathbf{ct})$
    $S_{\text{out}}[i_{\text{out}}] := (\mathbf{m}, \mathbf{ct})$
    $i_{\text{out}} := i_{\text{out}} + 1$
    return $\mathbf{ct}$

$\mathsf{EncSim}[\mathsf{pk}](\mathbf{m}; r)$
    $\mathbf{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{m}; r)$
    $S_{\text{in}}[i_{\text{in}}] := (\mathbf{m}, \mathbf{ct})$
    $i_{\text{in}} := i_{\text{in}} + 1$
    return $\mathbf{ct}$

$\mathsf{DecSim}(j)$
    return $S_{\text{out}}[j].\mathbf{m}$
```

$\mathcal{B}$ can perfectly simulate the $\mathcal{F}$-$\mathsf{IND\text{-}CPA^D}$ game for $\mathcal{A}$: $\mathcal{B}$'s simulated oracles for Enc and Eval are exactly equivalent to the true oracles in the $\mathcal{F}$-$\mathsf{IND\text{-}CPA^D}$ game, by construction; for the simulated Dec oracle, we have $\mathcal{O}_{\mathsf{DecSim}}(j) = S_{\text{out}}[j].\mathbf{m} = \mathsf{Eval}(F, S_{\text{in}}[i_1].\mathbf{m}, \ldots, S_{\text{in}}[i_k].\mathbf{m})$ for some F, i_1, …, i_k where F, i_1, …, i_k were submitted to $\mathcal{B}$'s simulated Eval oracle (with $F \in \mathcal{F}$), and where $S_{\text{in}}[i_1].\mathbf{m}$, …, $S_{\text{in}}[i_k].\mathbf{m}$ were submitted to $\mathcal{B}$'s simulated Enc oracle.

By $\mathcal{F}$-exactness of $\mathcal{E}$, it follows that

$$\begin{aligned}\mathcal{O}_{\mathsf{DecSim}}(j) &= \mathsf{Dec}_{\mathsf{sk}}(\mathsf{Eval}_{\mathsf{ek}}(F, S_{\text{in}}[i_1].\mathbf{ct}, \ldots, S_{\text{in}}[i_k].\mathbf{ct})) \\ &= \mathsf{Dec}_{\mathsf{sk}}(\mathsf{Eval}_{\mathsf{ek}}(F, \mathsf{Enc}_{\mathsf{pk}}(S_{\text{in}}[i_1].\mathbf{m}), \ldots, \mathsf{Enc}_{\mathsf{pk}}(S_{\text{in}}[i_k].\mathbf{m})),\end{aligned}$$

and $\mathcal{B}$ perfectly simulates the $\mathcal{F}$-$\mathsf{IND\text{-}CPA^D}$ decryption oracle for $\mathcal{A}$.

We note here that $\mathcal{B}$ needs to be able to check $F \in \mathcal{F}$ efficiently to be an efficient adversary overall. □

q-$\mathcal{F}$-IND-CPA$^\mathsf{D}$ from IND-CPA with Differential Privacy

Theorem 6.6 (following [Li+22, Theorem 2])
Let $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be an FHE scheme with a corresponding Estimate *such that $\tilde{\mathcal{E}} = (\mathcal{E}, \mathsf{Estimate})$ is statically approximate. Let M_t be a $\rho - KLDP$ mechanism as in [Li+22, Theorem 2]. Then for any PPT adversary $\mathcal{A}$ against the* q-$\mathcal{F}$-IND-CPA$^\mathsf{D}$*-security of $M[\tilde{\mathcal{E}}]$ there exists an adversary $\mathcal{B}$ against the* IND-CPA*-security of $\mathcal{E}$ such that*

$$\mathsf{Adv}^{\mathsf{q}\text{-}\mathcal{F}\text{-}\mathsf{IND\text{-}CPA^D}}[\mathcal{A}](\kappa) \leq \frac{q\rho}{2} + \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}[\mathcal{B}](\kappa)$$

Proof (informal, following [Li+22, Proof of Theorem 2]) The proof proceeds by game-hopping as follows: Game 0 is the q-$\mathcal{F}$-IND-CPA$^{\mathsf{D}}$ game with $M[\mathcal{E}]$, and Game 1 is the same game, except that the decryption oracle return a perturbed version of the stored plaintext. The difference in advantage between these two games is negligible, and can be bounded by $\frac{q\rho}{2}$ by using the property of the $\rho - KLDP$ mechanism. Game 1 is perfectly simulatable by $\mathcal{B}$ in its IND-CPA game, which concludes the proof. □

6.3 Robust FHE from VC with Input Privacy

The notions of VC and IND-$*$-security appear to offer similar guarantees, and to be related somehow. Intuitively, VC appears to provide IND-$*$-security (due to its input privacy notion), but to extend its guarantees to ensure the integrity of the computation as well. In this section, we will formalise this intuition by showing that any VC scheme secure in some adversarial setting is also IND-$*$-secure for the same adversarial model.

Definition 6.7 (FHE from publicly-delegatable VC)
Given any publicly-delegatable VC scheme $\mathcal{VC}$ (satisfying approximative correctness for all $f \in \mathcal{F}$) with PRIV-CCA1*-security, and such that keying material can be generated independently of any function f, we construct an FHE scheme $\mathcal{E}[\mathcal{VC}]$ as follows:*

```
KGen(1^κ)
    $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{VC}.\mathsf{Setup}(1^\kappa)$
    $\mathsf{ek} := \mathsf{pk}$
    return $(\mathsf{pk}, \mathsf{ek}, \mathsf{sk})$
```

```
Enc_pk(m)
    $(\sigma_x, \tau_x) \leftarrow \mathcal{VC}.\mathsf{ProbGen}_{\mathsf{key}}(\mathbf{m})$
    $\mathbf{ct} := \sigma_x \parallel \tau_x$
    return ct
```

```
Eval_ek(F, C_in)
    $\mathsf{pk}_F := (\mathsf{ek}, F)$
    parse $C_{in}$ as $(\sigma_i \parallel \tau_i)_{i=1}^m$
    $\sigma_y \leftarrow \mathcal{VC}.\mathsf{Compute}_{\mathsf{pk}_F}(F, x = \emptyset, w = C_{in})$
    return $(\tau_i)_{i=1}^m \parallel \sigma_y$
```

```
Dec_sk(ct)
    parse ct as $(\tau_i)_{i=1}^m \parallel \sigma_y$
    if $\mathcal{VC}.\mathsf{Verify}_{\mathsf{key}}(\tau_x, \sigma_y) = 1$
        $\mathbf{m} \leftarrow \mathcal{VC}.\mathsf{Decode}_{\mathsf{sk}}(\sigma_y)$
        return m
    else :
        return ⊥
```

Theorem 6.8 ($\mathcal{E}[\mathcal{VC}]$ is an IND-CCA1-secure FHE scheme)
$\mathcal{E}[\mathcal{VC}]$ from Definition 6.7 is compact with respect to the computation being evaluated, $\mathcal{F}$-approximate, and IND-CCA1*-secure. Additionally, if $\mathcal{VC}$ is perfectly $\mathcal{F}$-exact, then $\mathcal{E}[\mathcal{VC}]$ is also perfectly $\mathcal{F}$-exact.*

Proof $\mathcal{F}$-approximativity (respectively $\mathcal{F}$-exactness) of $\mathcal{E}[\mathcal{VC}]$ follows directly from the approximativity (respectively exactness) of $\mathcal{VC}$.

For IND-CCA1-security, let us consider an efficient adversary $\mathcal{A}$ against $\mathcal{E}[\mathcal{VC}]$. We can construct an elementary wrapper $\mathcal{B}$ around $\mathcal{A}$, which can perfectly simulate the IND-CCA1 game using its PRIV-CCA1 oracles, and $\mathcal{B}$ wins its game if and only if $\mathcal{A}$ wins its game. $\square$

We note that in order to re-frame the VC scheme into the FHE formalism, we require the keys pk_f and sk_f to be generated independently of f. For the FHE keys, this is achievable either by using bootstrapping, or by only consider functions f up to a certain multiplicative depth, and generating keys accordingly. If $\mathcal{VC}$ makes use of a proving system, this system should also support proving statements for arbitrary circuits, in order to ensure that $\mathcal{E}$ conforms with the non-interactivity requirement of FHE. We also note that the FHE scheme defined above is compact with respect to the computation f, but not compact with respect to the number of inputs to the computation. This limitation is also present in other IND-CCA1-constructions [Can+17].

Theorem 6.8 can be straightforwardly generalised to prove that $\mathcal{E}[\mathcal{VC}]$ is IND-ATK-secure if $\mathcal{VC}$ is PRIV-ATK-secure. We also note that (to the best of our knowledge), the generic construction from Definition 6.7 is the first to propose an IND-CCA-secure approximate scheme.

6.4 VC from Robust FHE

In this section, we investigate the connection between verifiable computation schemes and IND-CCA1-secure schemes in the other direction: we sketch a generic construction to augment an IND-CCA1 FHE scheme into a VC scheme, and we show that existing IND-CCA1-secure constructions in the literature are actually VC schemes.

Conceptually, re-shaping an IND-CCA1-secure FHE scheme into the VC formalism immediately yields a correct and input-private VC scheme. However, security is not achieved immediately, and one would to augment this scheme in order to provide security. In practice, we could use the compute-then-prove paradigm, and require that a proof of correct computation be adjoined to each ciphertext. We abstain from formalizing this construction here, as we will present a similar generic construction in Section 7.1.2, and prove its security by only requiring the weaker property that $\mathcal{E}$ be $\mathcal{F}$-IND-CPA$^{\mathsf{D}}$-secure (instead of IND-CCA1-secure).

We note here that some IND-CCA1-constructions for FHE (in particular the ones making use of the Naor-Yung paradigm), already provide a VC scheme (when instantiated with a sufficiently robust proof system). The Naor-Yung paradigm [NY90] is a generic technique to transform an IND-CPA-secure public key encryption scheme into a IND-CCA1-secure one. This is done by encrypting the same plaintext as two ciphertexts under two independent public keys, together with a proof that both ciphertexts encrypt the same plaintext. Canetti et al. introduced an IND-CCA1-secure FHE scheme by adapting this paradigm for FHE [Can+17]. Here, a proof of correct computation is needed during evaluation.

This approach is similar to our generic approach presented in the next chapter, but more inefficient (due to the doubling in the number of ciphertexts and in the size of the statements to be proven). Canetti et al.'s construction actually provides a maliciously-secure VC (for a suitable choice of zero-knowledge proof systems). We omit a formal proof, as it is very similar to the one we provide for our generic scheme.

This insight also allows us to demarcate this approach to IND-CCA1-secure FHE [Can+17] from other IND-CCA1-secure constructions [Bon+07; WWX18] (which rely on identity-based encryption and indistinguishability obfuscation, respectively); indeed, constructing a VC scheme from these other approaches seems to require adding proofs of computation in addition to the already heavy cryptographic machinery used in these other approaches, which does not seem to promise an efficient construction.

Chapter 7

Generic Robust VC Constructions for FHE

In this chapter, we provide generic constructions to achieve the strong FHE integrity notions from Chapter 5. We build our constructions from the ground up, striving to use the most simple primitives, in order to understand exactly what assumptions are needed to achieve FHE integrity. This allows us to modularly reason about our constructions, to provide simpler proofs, and for easier future optimizations on the level of the primitives. We start with a generic construction that achieves our standard definition of maliciously-secure VC, and extend it incrementally to support more expressive integrity notions.

7.1 Robust Non-Deterministic VC for Approximate FHE

7.1.1 Building Blocks

Before presenting our generic solution, we first review Succinct Non-interactive ARguments of Knowledge (SNARKs):

Definition 7.1 (SNARK)
Let $\mathcal{R}$ be an efficiently computable binary relation which consists of pairs of the form (x, w), where x is a statement, and w is a witness. Let L be the language associated with the relation $\mathcal{R}$, i.e., $L = \{x \mid \exists w.\mathcal{R}(x, w) = 1\}$.

A triple of polynomial time algorithms $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is a SNARK for an NP relation $\mathcal{R}$, if the following properties are satisfied:

Completeness. *For every true statement for the relation $\mathcal{R}$, an honest prover with a valid witness always convinces the verifier:*

$$\forall (x, w) \in \mathcal{R} : \ \Pr\left[\mathsf{Verify}_{\mathsf{vk}}(x, \pi) = 1 \ \middle| \ \begin{matrix} (\mathsf{crs}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^{\kappa}) \\ \pi \leftarrow \mathsf{Prove}_{\mathsf{crs}}(x, w) \end{matrix} \right] = 1$$

Knowledge Soundness. *For every PPT adversary, there exists a PPT extractor that gets full access to the adversary's state (including its random coins and inputs). Whenever the adversary produces a valid argument, the extractor can compute a witness with high probability:*

$$\forall \mathcal{A}\ \exists \mathcal{E} :\ \Pr\left[\begin{array}{c} \mathsf{Verify}_{\mathsf{vk}}(\tilde{x}, \tilde{\pi}) = 1 \\ \wedge \mathcal{R}(\tilde{x}, w') = 0 \end{array} \middle| \begin{array}{c} (\mathsf{crs}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\kappa) \\ ((\tilde{x}, \tilde{\pi}); w') \leftarrow \mathcal{A}|\mathcal{E}(\mathsf{crs}) \end{array}\right] = \mathsf{negl}(\kappa)$$

We stress here that this definition requires a non-black-box *extractor, i.e., the extractor gets full access to the adversary's state.*

Succinctness. *For any x and w, the length of the proof is given by $|\pi| = \mathsf{poly}(\kappa) \cdot \mathsf{polylog}(|x| + |w|)$.*

In our construction, we will require the *zero-knowledge* property in order to hide the server's input:

Definition 7.2 (zk-SNARK)
A zk-SNARK for a relation $\mathcal{R}$ is a SNARK for $\mathcal{R}$ with the following additional property:

Zero-Knowledge. *There exists a PPT simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that $\mathcal{S}_1$ outputs a simulated CRS crs and a trapdoor td; On input crs, x, and td, $\mathcal{S}_2$ outputs a simulated proof π, and for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, such that*

$$\left| \begin{array}{l} \Pr\left[\begin{array}{c} (x,w) \in \mathcal{R} \\ \wedge \\ \mathcal{A}_2(\pi) = 1 \end{array} \middle| \begin{array}{c} (\mathsf{crs}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\kappa) \\ (x, w) \leftarrow \mathcal{A}_1(1^\kappa, \mathsf{crs}) \\ \pi \leftarrow \mathsf{Prove}_{\mathsf{crs}}(x, w) \end{array}\right] - \\ \Pr\left[\begin{array}{c} (x,w) \in \mathcal{R} \\ \wedge \\ \mathcal{A}_2(\pi) = 1 \end{array} \middle| \begin{array}{c} (\mathsf{crs}', \mathsf{td}) \leftarrow \mathcal{S}_1(1^\kappa) \\ (x, w) \leftarrow \mathcal{A}_1(1^\kappa, \mathsf{crs}') \\ \pi \leftarrow \mathcal{S}_2(\mathsf{crs}', \mathsf{td}, x) \end{array}\right] \end{array} \right| = \mathsf{negl}(\kappa)$$

We now recall a somewhat newer and stronger notion for SNARKs (due to Fiore and Nitulescu) [FN16], namely *O-SNARKs.* For the knowledge soundness of classical (zk-)SNARKs, the extractor is a non-black-box algorithm with the same input as the prover. When used as part of a broader protocol, a (potentially) malicious prover might have access to additional information, e.g., through some additional input, or through oracles. In these situations, an extractor as defined in Definition 7.1 might not exist. The following definition of O-SNARKs models this setting and captures this limitation, by requiring that knowledge soundness hold for adversaries with access to auxiliary information and an oracle O sampled from an oracle family $\mathbb{O}$.

Definition 7.3 (O-SNARK [FN16])
An O-SNARK for a relation $\mathcal{R}$ satisfies adaptive knowledge soundness with respect to the oracle family $\mathbb{O} = \{O\}$ if for all auxiliary information $z \leftarrow \mathcal{Z}$, for every PPT adversary $\mathcal{A}$, there exists a PPT extractor $\mathsf{Ext}_{\mathcal{A}}$ such that

$$\Pr\left[\begin{array}{c|c} & (\mathsf{crs}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\kappa) \\ \mathsf{Verify}_{\mathsf{vk}}(x, \pi) = 1 & O \leftarrow \mathbb{O} \\ \wedge\, (x, w) \notin \mathcal{R} & (x, \pi) \leftarrow \mathcal{A}^O(\mathsf{crs}, z) \\ & w \leftarrow \mathsf{Ext}_{\mathcal{A}}(\mathsf{crs}, z, \mathsf{qt}) \end{array}\right] = \mathsf{negl}(\kappa),$$

where $\mathsf{qt} = \{q_i, O(q_i)\}_i$ is the transcript of all oracle queries and answers made and received by $\mathcal{A}^O$ during its execution.

7.1.2 Maliciously-Secure VC

Let us now define the two following relations and corresponding O-SNARKs:

Definition 7.4 (FHE Integrity Relation)
Let $\mathcal{R}_F$ capture correct evaluations of a function F as follows:

$$\mathcal{R}_F = \left\{ \left((\mathbf{ct}_{out}, C_{in}), C'_{in}\right) \middle| \begin{array}{c} \mathbf{ct}_{out} = \mathsf{Eval}_{\mathsf{ek}}(F, C_{in}, C'_{in}) \\ \wedge\, \forall i : C_{in}[i] \in \Omega \\ \wedge\, \forall i : C'_{in}[i] \in \Omega \end{array} \right\},$$

where Ω denotes the set of all valid ciphertexts with respect to the public key pk.

The check for $C'_{\text{in}}[i] \in \Omega$ can be implemented either by showing that $C'_{\text{in}}[i]$ is the (deterministic) embedding in cipherspace of a valid plaintext, or by showing that $C'_{\text{in}}[i]$ is a valid encryption of a valid plaintext. For the former, the check could be implemented as $C'_{\text{in}}[i]_0 \in \mathcal{M} \wedge C'_{\text{in}}[i]_1 = 0$ (i.e., $C'_{\text{in}}[i]$ is the standard BGV embedding of a valid plaintext), while for the latter we could for example use $\Pi_{\mathsf{Enc}}.\mathsf{Verify}_{\mathsf{vk}}(C'_{\text{in}}[i]; \mathbf{m}, r)$ by relying on another SNARK for the relation $\mathcal{R}_{\mathsf{Enc}} = \left\{(\mathbf{ct}, (\mathbf{m}, r)) \,\middle|\, \mathbf{m} \in \mathcal{M} \wedge r \in \mathcal{R}_{\mathcal{M}} \wedge \mathbf{ct} = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{m}; r)\right\}$.

The check for $C_{\text{in}}[i] \in \Omega$ can be implemented similarly, but can also be simplified if the encryptor is a trusted party: instead of proving that $C_{\text{in}}[i]$ fulfils some validity property, we only check that it is in some set of honestly generated ciphertexts. This is especially useful when the verifier and the encryptor are the same party, as it can trivially perform this check by matching $C_{\text{in}}[i]$ against the ciphertexts it sent to the server for evaluation. In the following, let Π_F be an O-SNARK for $\mathcal{R}_F$.

We also make use of the following definition in the proofs:

Definition 7.5 (Recomputation from Extracted Witness)
For $\mathcal{R}_F$ as defined above, we define a corresponding function $\mathsf{Recompute}_{\mathsf{pk},\mathsf{ek}}$ *that takes a witness w as input, and returns a ciphertext $\mathbf{ct}_{out}$:*

```
$\underline{\mathsf{Recompute}_{\mathsf{pk},\mathsf{ek}}(C_{in}, C'_{in})}$

$\mathbf{ct}_{out} := \mathcal{E}.\mathsf{Eval}_{\mathsf{ek}}(F, C_{in}, C'_{in})$
**return** $\mathbf{ct}_{out}$
```

Note that by construction, we have that for all $\mathbf{ct}_{out}$ and w, $((\mathbf{ct}_{out}, C_{in}), w) \in \mathcal{R} \iff \mathbf{ct}_{out} = \mathsf{Recompute}_{\mathsf{pk},\mathsf{ek}}(C_{in}, w)$.

We are now ready to introduce our first generic construction for maliciously-secure VC:

Definition 7.6 (Generic Construction)
For an FHE scheme $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ and the SNARK from Definition 7.4, we define a new FHE scheme $\mathcal{VC}[\mathcal{E}]$ as follows:

$\mathsf{Setup}(1^\kappa), \mathsf{KGen}_{\mathsf{pk}}(f)$: *$\mathcal{VC}[\mathcal{E}]$ generates public and secret keys for the FHE scheme $\mathcal{E}$ and for the SNARKs;*

$\mathsf{ProbGen}_{\mathsf{pk}}(x)$: *both σ_x and τ_x consists of encryptions of the plaintexts in x together with proofs of their correct encryption;*

$\mathsf{Compute}_{\mathsf{pk}_f}(\sigma_x, w)$: *σ_y consists of the evaluation of f with inputs from σ_x and w, as well as a proof that i) verifies the encryption proofs in σ_x, and ii) certifies that the evaluation of f was done correctly;*

$\mathsf{Verify}_{\mathsf{sk}_f}(\tau_x, \sigma_y)$: *the verification step simply uses Π_f to check if the proof included in σ_y is correct;*

$\mathsf{Decode}_{\mathsf{sk},\mathsf{sk}_f}(\sigma_y)$: *this is simply the FHE decryption.*

We define all steps of $\mathcal{VC}[\mathcal{E}]$ more formally below.

```
$\mathcal{VC}[\mathcal{E}]$.Setup$(1^\kappa)$
    $(\mathsf{pk}^E, \mathsf{ek}^E, \mathsf{sk}^E) \leftarrow \mathcal{E}.\mathsf{KGen}(1^\kappa)$
    $\mathsf{pk} := (\mathsf{pk}^E, \mathsf{ek}^E, \mathsf{crs}_{\mathsf{Enc}})$
    $\mathsf{sk} := (\mathsf{sk}^E, \mathsf{vk}_{\mathsf{Enc}})$
    return $(\mathsf{pk}, \mathsf{sk})$
```

```
$\mathcal{VC}[\mathcal{E}]$.KGen$_{\mathsf{pk}}(f)$
    $(\mathsf{crs}_f, \mathsf{vk}_f) \leftarrow \Pi_f.\mathsf{Setup}(1^\kappa)$
    $\mathsf{pk}_f := (\mathsf{crs}_f, \mathsf{pk}.\mathsf{ek}^E)$
    $\mathsf{sk}_f := \mathsf{vk}_f$
    return $(\mathsf{pk}_f, \mathsf{sk}_f)$
```

```
$\mathcal{VC}[\mathcal{E}]$.ProbGen$_{\mathsf{pk}}(x)$
    parse $\mathsf{pk}$ as $(\mathsf{pk}^E, \mathsf{ek}^E)$
    parse $x$ as $M_{in}$
    for $\mathbf{m}_i$ in $M_{in}$ :
        $r_i \leftarrow \mathcal{R}_{\mathcal{M}}$
        $\mathbf{ct}_i \leftarrow \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(\mathbf{m}_i; r_i)$
    $\sigma_x := C_{in}$
    $\tau_x := \sigma_x$
    return $(\sigma_x, \tau_x)$
```

```
$\mathcal{VC}[\mathcal{E}]$.Compute$_{\mathsf{pk}_f}(\sigma_x, w)$
    parse $\mathsf{pk}_f$ as $(\mathsf{crs}_f, \mathsf{pk}.\mathsf{ek}^E)$
    parse $\sigma_x$ as $C_{in}$
    parse $w$ as $C'_{in}$
    $\mathbf{ct}_{out} \leftarrow \mathcal{E}.\mathsf{Eval}_{\mathsf{ek}}(F, C_{in}, C'_{in})$
    $\pi \leftarrow \Pi_f.\mathsf{Prove}_{\mathsf{crs}_f}((\mathbf{ct}_{out}, C_{in}); C'_{in})$
    $\sigma_y := (\mathbf{ct}_{out}, \pi)$
    return $\sigma_y$
```

```
$\mathcal{VC}[\mathcal{E}]$.Verify$_{\mathsf{sk}_f}(\tau_x, \sigma_y)$
    parse $\tau_x$ as $C_{in}$
    parse $\sigma_y$ as $(\mathbf{ct}_{out}, \pi)$
    $b \leftarrow \Pi_f.\mathsf{Verify}_{\mathsf{vk}_f}((\mathbf{ct}_{out}, C_{in}), \pi)$
    return $b$
```

```
$\mathcal{VC}[\mathcal{E}]$.Decode$_{\mathsf{sk},\mathsf{sk}_f}(\sigma_y)$
    parse $\sigma_y$ as $(\mathbf{ct}_{out}, \pi)$
    $\mathbf{m} \leftarrow \mathcal{E}.\mathsf{Dec}_{\mathsf{sk}^E}(\mathbf{ct}_{out})$
    return $\mathbf{m}$
```

Theorem 7.7 ($\mathcal{VC}[\mathcal{E}]$ is Correct)
Let $\mathcal{E}$ be an $\mathcal{F}$-approximate FHE scheme, and let Π and $\mathcal{VC}[\mathcal{E}]$ be as in Definition 7.6. $\mathcal{VC}[\mathcal{E}]$ is correct.

Proof Let $f \in \mathcal{F}$, and let $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\kappa)$, $(\mathsf{pk}_f, \mathsf{sk}_f) \leftarrow \mathsf{KGen}(f)$, $(\sigma_x, \tau_x) \leftarrow \mathsf{ProbGen}_{\mathsf{key}}(x)$, and $\sigma_y \leftarrow \mathsf{Compute}_{\mathsf{pk}_f}(\sigma_x, w)$.

$\mathsf{Verify}_{\mathsf{key}}(\tau_x, \sigma_y) = 1$ by construction and by the (perfect) correctness of Π, and $\left\|\mathsf{Decode}_{\mathsf{sk},\mathsf{sk}_f}(\sigma_y, w) - f(x, w)\right\| \leq \epsilon$ by construction and by $\mathcal{F}$-approximativity of $\mathcal{E}$. □

Theorem 7.8 ($\mathcal{VC}[\mathcal{E}]$ is PRIV-CCA1-secure)
Let $\mathcal{E}$ be an $\mathcal{F}$-IND-CPA$^{\mathsf{D}}$-secure FHE scheme, and let Π and $\mathcal{VC}[\mathcal{E}]$ be as in Definition 7.6. $\mathcal{VC}[\mathcal{E}]$ is PRIV-CCA1-secure.

In particular, for every PRIV-CCA1-adversary $\mathcal{A}$ for Π, there exists an $\mathcal{F}$-IND-CPA$^{\mathsf{D}}$-adversary $\mathcal{B}$ for $\mathcal{E}$, and a KS-adversary $\mathcal{C}$ against Π such that

$$\mathsf{Adv}^{\mathsf{PRIV\text{-}CCA1}}[\mathcal{A}](\kappa) \leq \mathsf{Adv}^{\mathcal{F}\text{-}\mathsf{IND\text{-}CPA}^{\mathsf{D}}}[\mathcal{B}](\kappa) + \mathsf{Adv}^{\mathsf{KS}}[\mathcal{C}](\kappa)$$

Proof Informally, $\mathcal{VC}[\mathcal{E}]$ allows us to reduce the PRIV-CCA1 security to the $\mathcal{F}$-IND-CPA$^{\mathsf{D}}$-security of $\mathcal{E}$, by only allowing for decryptions of ciphertexts that have been computed by evaluating a function $f \in \mathcal{F}$ on honestly encrypted inputs, which is enforced through the proof schemes.

```
Game 1:
$b \leftarrow\!\!\$ \{0,1\}$
$(\mathsf{pk}^E, \mathsf{sk}^E) \leftarrow \mathcal{E}.\mathsf{Setup}(1^\kappa)$
$(\mathsf{crs}^V, \mathsf{vk}^V, \mathsf{td}^V) \leftarrow \Pi.\mathsf{Setup}(1^\kappa)$
$\mathsf{pk} := (\mathsf{pk}^E, \mathsf{crs}^V)$
$(x_0, x_1) \leftarrow \mathcal{A}_1^{\mathcal{O}_1}(1^\kappa, \mathsf{pk})$
$r \leftarrow \mathcal{R}_{\mathcal{M}}$
$\sigma_b^* := \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(\mathbf{m}_b; r)$
$\tau_b^* := \sigma_b^*$
$\widehat{b} \leftarrow \mathcal{A}_2^{\mathcal{O}_2}(\sigma_b^*, \tau_b^*)$
return $b = \widehat{b}$

$\mathcal{O}_{\mathsf{ProbGen}}(x)$
$r \leftarrow \mathcal{R}_{\mathcal{M}}$
$\sigma_x := \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(x; r)$
$\tau_x := \sigma_x$
return $(\sigma_x, \tau_x)$

$\mathcal{O}_{\mathsf{Decode}}(\sigma_y, \tau_x)$
$a \leftarrow \mathsf{Verify}_{\mathsf{key}}(\sigma_y)$
if $a = 0$ :
  return $\perp$
else :
  $\mathbf{m} \leftarrow \mathcal{E}.\mathsf{Dec}_{\mathsf{sk}}(\mathbf{ct})$
  return $\mathbf{m}$
```

```
Game 2:
$b \leftarrow\!\!\$ \{0,1\}$
$(\mathsf{pk}^E, \mathsf{sk}^E) \leftarrow \mathcal{E}.\mathsf{Setup}(1^\kappa)$
$(\mathsf{crs}^V, \mathsf{vk}^V, \mathsf{td}^V) \leftarrow \Pi.\mathsf{Setup}(1^\kappa)$
$\mathsf{pk} := (\mathsf{pk}^E, \mathsf{crs}^V)$
$(x_0, x_1) \leftarrow \mathcal{A}_1^{\mathcal{O}_1}(1^\kappa, \mathsf{pk})$
$r \leftarrow \mathcal{R}_{\mathcal{M}}$
$\sigma_b^* := \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(\mathbf{m}_b; r)$
$\tau_b^* := \sigma_b^*$
$\widehat{b} \leftarrow \mathcal{A}_2^{\mathcal{O}_2}(\sigma_b^*, \tau_b^*)$
return $b = \widehat{b}$

$\mathcal{O}_{\mathsf{ProbGen}}(x)$
$r \leftarrow \mathcal{R}_{\mathcal{M}}$
$\sigma_x := \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(x; r)$
$\tau_x \leftarrow \Pi.\mathsf{Prove}_{\mathsf{crs}}(\sigma^*; x, r)$
return $(\sigma_x, \tau_x)$

$\mathcal{O}_{\mathsf{Decode}}(\sigma_y, \tau_x)$
$a \leftarrow \mathsf{Verify}_{\mathsf{key}}(\sigma_y)$
if $a = 0$ :
  return $\perp$
else :
  $w \leftarrow \mathsf{Ext}_{\mathcal{A}}(\mathsf{crs}_f, (\sigma_y, \tau_x), \mathsf{qt})$
  $\sigma_y' := \mathsf{Recompute}_{\mathsf{pk},\mathsf{pk}_f}(\tau_x, w)$
  $\mathbf{m} \leftarrow \mathcal{E}.\mathsf{Dec}_{\mathsf{sk}}(\sigma_y')$
  return $\mathbf{m}$
```

$\mathcal{O}_{\mathsf{KGen}}$ is not shown, as it is not modified throughout the games. The oracles available to $\mathcal{A}$ are $\mathcal{O}_1 = (\mathcal{O}_{\mathsf{KGen}}, \mathcal{O}_{\mathsf{ProbGen}}, \mathcal{O}_{\mathsf{Decode}})$, and $\mathcal{O}_2 = (\mathcal{O}_{\mathsf{KGen}}, \mathcal{O}_{\mathsf{ProbGen}})$.

Figure 7.1: Games for the proof of Theorem 7.8. Gray lines highlight differences with the previous game.

More formally, consider the games in Figure 7.1. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary against the PRIV-CCA1 security of $\Phi[\mathcal{E}]$. We first construct a sequence of games for $\mathcal{A}$ (starting with the PRIV-CCA1 game for $\mathcal{VC}[\mathcal{E}]$), and show that $\mathcal{A}$'s advantage does not increase significantly from game to game. We can then construct an $\mathcal{F}$-IND-CPA$^{\mathsf{D}}$-adversary against $\mathcal{E}$ by making non-black-box use of $\mathcal{A}$. Let W_i be the probability that the adversary $\mathcal{A}$ wins Game i.

$G_0 \to G_1$: Game 0 is the VC-CCA1 input privacy game (Definition 5.2) for $\mathcal{VC}[\mathcal{E}]$, and $\Pr[W_0] = \mathsf{Adv}^{\mathsf{VC\text{-}CCA1}}[\mathcal{A}](\kappa)$. Game 1 is the same game as Game 0, except that we expanded the definitions of $\mathcal{VC}[\mathcal{E}].\mathsf{ProbGen}$, $\mathcal{VC}[\mathcal{E}].\mathsf{Verify}$ and $\mathcal{VC}[\mathcal{E}].\mathsf{Decode}$, hence $\Pr[W_0] = \Pr[W_1]$.

$G_1 \to G_2$: Going from Game 1 to Game 2, only the decryption oracle changes: instead of decrypting $\mathcal{A}$'s ciphertext, the challenger first verifies the proof, then recovers a witness w for the statement x, and returns the decryption of the *recomputed* ciphertext **ct**. We now argue that $|\Pr[W_1] - \Pr[W_2]| = \mathsf{negl}(\kappa)$: clearly, both games are the same until the decryption oracle queried on the same input σ returns different plaintexts in Game 1 and Game 2. Let E_{Dec} denote this event; by the difference lemma, we have $|\Pr[W_1] - \Pr[W_2]| \leq \Pr[E_{\mathsf{Dec}}]$.

$\Pr[E_{\mathsf{Dec}}]$ is negligible, as otherwise we would have an adversary against the knowledge soundness (with oracles $\mathcal{O}_1$ and additional data pk) of the proving scheme:

$$\begin{aligned}
\Pr[E_{\mathsf{Dec}}] &= \Pr\left[\begin{array}{c} \Pi.\mathsf{Verify}_{\mathsf{vk}}(\tau_x, \sigma_y) = 1 \\ \wedge\, \sigma'_y = \mathsf{Recompute}_{\mathsf{pk},\mathsf{pk}_f}(\tau_x, w) \\ \wedge\, \mathcal{E}.\mathsf{Dec}_{\mathsf{sk}}(\sigma_y) \neq \mathcal{E}.\mathsf{Dec}_{\mathsf{sk}}(\sigma'_y) \end{array} \middle| \begin{array}{l} (\mathsf{crs}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\kappa) \\ (x, \pi) \leftarrow \mathcal{A}_1^{\mathcal{O}_1}(\mathsf{crs}, \mathsf{pk}) \\ w \leftarrow \mathsf{Ext}_{\mathcal{A}_1}(\mathsf{crs}, \mathsf{pk}, \mathsf{qt}) \end{array}\right] \\
&\leq \Pr\left[\begin{array}{c} \Pi.\mathsf{Verify}_{\mathsf{vk}}(\tau_x, \sigma_y) = 1 \\ \wedge\, \sigma_y \neq \mathsf{Recompute}_{\mathsf{pk},\mathsf{pk}_f}(\tau_x, w) \end{array} \middle| \begin{array}{l} (\mathsf{crs}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\kappa) \\ (x, \pi) \leftarrow \mathcal{A}_1^{\mathcal{O}_1}(\mathsf{crs}, \mathsf{pk}) \\ w \leftarrow \mathsf{Ext}_{\mathcal{A}_1}(\mathsf{crs}, \mathsf{pk}, \mathsf{qt}) \end{array}\right] \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\mathsf{Dec}_{\mathsf{sk}}(\cdot) \text{ is injective}) \\
&\leq \Pr\left[\begin{array}{c} \Pi.\mathsf{Verify}_{\mathsf{vk}}(\tau_x, \sigma_y) = 1 \\ \wedge\, (x, w) \notin R \end{array} \middle| \begin{array}{l} (\mathsf{crs}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\kappa) \\ (x, \pi) \leftarrow \mathcal{A}_1^{\mathcal{O}_1}(\mathsf{crs}, \mathsf{pk}) \\ w \leftarrow \mathsf{Ext}_{\mathcal{A}_1}(\mathsf{crs}, \mathsf{pk}, \mathsf{qt}) \end{array}\right] \quad (\text{Def. } 7.5) \\
&= \mathsf{Adv}^{\mathsf{KS}}[\mathcal{A}_1](\kappa)
\end{aligned}$$

Going through all games in sequence, we showed $|\Pr[W_0] - \Pr[W_2]| = \mathsf{negl}(\kappa)$.

We can now construct an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ against $\mathcal{F}$-IND-CPA$^{\mathsf{D}}$ of $\mathcal{E}$ by using $\mathcal{A}$, simulating Game 2 with the help of the $\mathcal{F}$-IND-CPA$^{\mathsf{D}}$ oracles. $\mathcal{B}$ proceeds as follows:

- On input $(1^\kappa, \mathsf{pk}, \mathsf{ek})$ (for the $\mathcal{F}$-IND-$\mathsf{CPA}^{\mathsf{D}}$ game against $\mathcal{E}$), $\mathcal{B}$ samples $(\mathsf{crs}, \mathsf{vk}) \leftarrow \Pi.\mathsf{KGen}(1^\kappa)$ and sets $\mathsf{pk}' = (\mathsf{pk}, \mathsf{crs})$.
- $\mathcal{B}$ then runs $\mathcal{A}(1^\kappa, \mathsf{pk}', \mathsf{ek})$ in a non-black-box manner as follows:
  - Whenever $\mathcal{A}$ issues a decryption query for $\mathbf{ct}'$, $\mathcal{B}$ first checks whether $\mathcal{VC}[\mathcal{E}].\mathsf{Verify}_{\mathsf{vk}}(\mathbf{ct}') = 1$. If this is not the case, $\mathcal{B}$ can safely answer $\mathcal{A}$'s query with $\perp$. If the verification check passes, $\mathcal{B}$ parses $\mathbf{ct}'$ as (x, π), and retrieves a corresponding witness w using the extractor $\mathcal{E}$. $\mathcal{B}$ then computes $j = \overline{\mathsf{Recompute}}_{\mathsf{pk},\mathsf{ek}}(w)$, and submits j to its decryption oracle to get $\mathbf{m}$, which it returns to $\mathcal{A}$.
  - Whenever $\mathcal{A}$ asks for the challenge ciphertext for the pair $(\mathbf{m}_0, \mathbf{m}_1)$, $\mathcal{B}$ relays this pair to its own challenger, and receives back σ^*. $\mathcal{B}$ then simulates a proof π^* for σ^* (the statement is in the language, but $\mathcal{B}$ does not know a witness), and sends $(\sigma^*, \mathbf{m}^*)$ to $\mathcal{A}$.
  - When $\mathcal{A}$ outputs $\widehat{b}$, $\mathcal{B}$ also outputs $\widehat{b}$.

```
Game 2:
$b \leftarrow\!\$ \{0,1\}$
$(\mathsf{pk}^E, \mathsf{sk}^E) \leftarrow \mathcal{E}.\mathsf{Setup}(1^\kappa)$
$(\mathsf{crs}^V, \mathsf{vk}^V, \mathsf{td}^V) \leftarrow \Pi.\mathsf{Setup}(1^\kappa)$
$\mathsf{pk} := (\mathsf{pk}^E, \mathsf{crs}^V)$
$(x_0, x_1) \leftarrow \mathcal{A}_1^{\mathcal{O}_1}(1^\kappa, \mathsf{pk})$
$r \leftarrow \mathcal{R}_\mathcal{M}$
$\sigma_b^* := \mathcal{E}.\mathsf{Enc}_\mathsf{pk}(\mathbf{m}_b; r)$
$\tau_b^* := \sigma_b^*$
$\widehat{b} \leftarrow \mathcal{A}_2^{\mathcal{O}_2}(\sigma_b^*, \tau_b^*)$
return $b = \widehat{b}$
```

```
$\mathcal{O}_\mathsf{ProbGen}(x)$
$r \leftarrow \mathcal{R}_\mathcal{M}$
$\sigma_x := \mathcal{E}.\mathsf{Enc}_\mathsf{pk}(x; r)$
$\tau_x := \sigma_x$
return $(\sigma_x, \tau_x)$
```

```
$\mathcal{O}_\mathsf{Decode}(\sigma_y, \tau_x)$
$a \leftarrow \mathsf{Verify}_\mathsf{key}(\sigma_y)$
if $a = 0$ :
    return $\perp$
else :
    $w \leftarrow \mathsf{Ext}_\mathcal{A}(\mathsf{crs}_f, (\sigma_y, \tau_x), \mathsf{qt})$
    $\sigma'_y := \mathsf{Recompute}_{\mathsf{pk},\mathsf{pk}_f}(\tau_x, w)$
    $\mathbf{m} \leftarrow \mathcal{E}.\mathsf{Dec}_\mathsf{sk}(\sigma'_y)$
    return $\mathbf{m}$
```

```
$\mathcal{B}_1[\mathcal{A}](1^\kappa, \mathsf{pk}^E)$
$(\mathsf{crs}^V, \mathsf{vk}^V, \mathsf{td}^V) \leftarrow \Pi.\mathsf{Setup}(1^\kappa)$
$\mathsf{pk} := (\mathsf{pk}^E, \mathsf{crs}^V)$
$(x_0, x_1) \leftarrow \mathcal{A}_1^{\mathcal{O}_1}(1^\kappa, \mathsf{pk})$
return $(\mathbf{m}_0, \mathbf{m}_1)$
```

```
$\mathcal{B}_2[\mathcal{A}](\sigma^*, \tau^*)$
$\widehat{b} \leftarrow \mathcal{A}_2(\sigma^*, \pi^*)$
return $\widehat{b}$
```

```
$\mathsf{ProbGenSim}(x)$
$r \leftarrow \mathcal{R}_\mathcal{M}$
$\sigma_x \leftarrow \mathcal{O}_\mathsf{Enc}(x; r)$
$\tau_x := \sigma_x$
return $(\sigma_x, \tau_x)$
```

```
$\mathsf{DecodeSim}(\sigma, \tau)$
$a \leftarrow \mathsf{Verify}_\mathsf{key}(\sigma_y)$
if $a = 0$ :
    return $\perp$
else :
    $w \leftarrow \mathsf{Ext}_\mathcal{A}(\mathsf{crs}_f, (\sigma_y, \tau_x), \mathsf{qt})$
    $j := \overline{\mathsf{Recompute}}_{\mathsf{pk},\mathsf{pk}_f}(\tau_x, w)$
    $\mathbf{m} \leftarrow \mathcal{O}_\mathsf{Dec}(j)$
    return $\mathbf{m}$
```

$\overline{\mathsf{Recompute}}$ performs the same computations as $\mathsf{Recompute}$, but uses the encryption and evaluation oracles from the $\mathcal{F}$-IND-CPA^D game instead of encrypting and evaluating directly. By running $\overline{\mathsf{Recompute}}(w)$, $\mathcal{B}$ ensures that **ct** is stored at the index j in the challenger's state S_out, and can thus recover **m** through a query to its $\mathcal{F}$-IND-CPA^D decryption oracle, and $\mathcal{B}$ can perfectly simulate Game 2 for $\mathcal{A}$.

$\square$

Theorem 7.9 ($\mathcal{VC}\ [\mathcal{E}]$ is VER-CCA1-secure)
Let $\mathcal{E}$ be an FHE scheme, and let Π and $\mathcal{VC}\ [\mathcal{E}]$ be as in Definition 7.6. $\mathcal{VC}\ [\mathcal{E}]$

is $\mathsf{VER\text{-}CCA1}$*-secure.*

In particular, for every $\mathsf{VER\text{-}CCA1}$*-adversary* $\mathcal{A}$ *for* Π*, there exists a* KS*-adversary* $\mathcal{B}$ *against* Π_f *such that*

$$\mathsf{Adv}^{\mathsf{VER\text{-}CCA1}}[\mathcal{A}](\kappa) = \mathsf{Adv}^{\mathsf{KS}}[\mathcal{B}](\kappa)$$

Proof Let $\mathcal{A}$ be a PPT adversary against the $\mathsf{VER\text{-}CCA1}$-security of Π. $\mathcal{B}$ simulates the KS game for $\mathcal{A}$, and recovers x, f, and σ_y such that $\Pi_f.\mathsf{Verify}_{\mathsf{vk}_f}(\tau_x, \sigma_y) = 1$ and $(\sigma_y, (\tau_x, w)) \notin \mathcal{R}_f$ (for $(\sigma_x, \tau_x) \leftarrow \mathsf{ProbGen}_{\mathsf{pk}}(x)$) with some non-negligible probability, and $\mathcal{B}$ wins the KS game exactly when $\mathcal{A}$ wins in the $\mathsf{VER\text{-}CCA1}$-game.

We note that in all these proofs, we did not require that $\mathcal{E}$ be exact. In fact, our construction can transform any $\mathcal{F}$-$\mathsf{IND\text{-}CPA}^{\mathsf{D}}$-secure scheme (including the DP-augmented CKKS scheme [Li+22]) into a $\mathsf{IND\text{-}CCA1}$-secure VC scheme. To the best of our knowledge, this is the first construction to achieve this. We also note that our construction is conceptually simpler than the one from [Boi+21], which includes some parts that are required by the concrete approach chosen by Bois et al. (e.g., the use of homomorphic hashing, or the use of a commit-and-prove scheme). Additionally, we explicitly take into account oracles and additional information available to a malicious prover using the O-SNARK formalism, in order to faithfully capture a strong and realistic adversary; to the best of our knowledge, we are the first to do so in the VC literature for FHE.

7.1.3 Maliciously-Secure VC with Server Context-Hiding

However, this first generic construction does not even offer server context hiding, as the noise present in σ_y could leak information about w. In order to achieve weak context hiding, we make use of a *circuit privacy mechanism* to make the noise independent of w. We recall the definition of circuit privacy:

Definition 7.10 (Circuit Privacy)
Formally, an FHE scheme $\mathcal{E}$ *is* circuit-private *if there exists a simulator* Sim *such that for all ciphertext circuits* F *(and corresponding plaintext circuits* f*) and for all* M_{in} *in the domain of* f*, the distributions* $\mathcal{E}.\mathsf{Sim}(\mathsf{pk}, f(x))$ *and* $\mathcal{E}.\mathsf{Eval}_{\mathsf{ek}}(f, C_{in})$ *are statically indistinguishable, given* $(\mathsf{pk}, \mathsf{ek}, \mathsf{sk}) \leftarrow \mathcal{E}.\mathsf{KGen}(1^\kappa)$, $\forall i : C_{in}[i] \leftarrow \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(M_{in}[i])$.

In practice, circuit privacy can be implemented in two different ways. The first approach uses *noise flooding* and was introduced by Gentry in [Gen09, Chapter 21]. Here, the noise in the output ciphertext is "drowned" under a much larger amount of noise (e.g., by adding a random sum of honestly generated encryptions of 0, as proposed in [FNP20; Boi+21]. A second approach by Ducas and Stehlé uses repeated applications of the bootstrapping procedure to attain the independence of the resulting noise with respect to the computation's inputs. Let us now show that the generic VC scheme from Definition 7.6 instantiated with a circuit-private FHE scheme achieves server context-hiding. We note that we do not rely on any specific circuit privacy mechanism (contrary to [FNP20; Boi+21]) in the following theorem and proof.

Theorem 7.11
Let $\mathcal{VC}$ be the generic VC scheme from Definition 7.6, where $\mathcal{E}$ is a circuit-private, $\mathcal{F}$-$\mathsf{IND\text{-}CPA^D}$-secure FHE scheme. Then $\mathcal{VC}$ is a maliciously-secure VC with correctness, input hiding, security, and server context hiding.

Proof Correctness, input hiding, and security follow directly from Proofs 4 to 6.

In order to prove that $\mathcal{VC}$ is server context hiding, we simply define the simulators S_1, S_2, and S_3 from Definition 5.4:

$S_1(1^\kappa) \to (\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{td}):$ S_1 simply runs $\mathsf{Setup}(1^\kappa)$, and outputs an empty trapdoor td.

$S_2(\mathsf{td}, f) \to (\mathsf{pk}_f^*, \mathsf{sk}_f^*, \mathsf{td}_f):$ S_2 proceeds as $\mathsf{KGen}_{\mathsf{pk}}(f)$, but uses the simulated keys $(\mathsf{crs}_f, \mathsf{vk}_f, \mathsf{td}_f) \leftarrow \Pi.\mathsf{SetupSim}(1^\kappa)$ instead.

$S_3(\mathsf{td}_f, \tau_x, f(x,w) \to \sigma_y^*:$ s_3 first computes a fake result ciphertext $\mathbf{ct}_{\text{out}}^* \leftarrow \mathcal{E}.\mathsf{Sim}(\mathsf{pk}, f(x,w))$ by using $\mathcal{E}$'s circuit privacy simulator, and then generates a simulated proof π^* for the statement $(\mathbf{ct}_{\text{out}}^*, C_{\text{in}})$ using the SNARK simulator and the trapdoor td_f.

S_1 obviously generates the same distribution as Setup. S_2 also generates a distribution that is indistinguishable from KGen due to the zero-knowledge property of the SNARK. Finally, S_3 generates $\sigma_y^* = (\mathbf{ct}_{\text{out}}^*, \pi^*)$ indistinguishable from σ_y, owing to the circuit privacy of $\mathcal{E}$ and the zero-knowledge property of the SNARK. $\square$

7.1.4 Maliciously-Secure VC with Full Context-Hiding

The server context-hiding construction from above does not satisfy full context-hiding, as an adversary with access to the secret key sk would be able to decrypt $\tau_x = C_{\text{in}}$ to recover x. In order to remediate this, τ_x needs to retain information about x, but it should hide from an adversary with the FHE secret key. We will realise this property by using a commitment scheme to commit to x in τ_x.

Let us recall the definition and properties of a commitment scheme:

Definition 7.12 (Commitment)
A commitment scheme $\mathsf{Com} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ *is a triple of PPT algorithms:*

$\mathsf{Setup}(1^\kappa) \to \mathsf{ck}:$ *generates a public key* ck*;*

$\mathsf{Commit}_{\mathsf{ck}}(m) \to (\mathsf{com}, o):$ *from a message* m*, generates a commitment* com *and a corresponding opening* o*;*

$\mathsf{Open}_{\mathsf{ck}}(\mathsf{com}, m, o) \to 0/1:$ *outputs 1 if* (m, com, o) *is a valid message-commitment-opening tuple, and 0 otherwise.*

A commitment scheme is secure if it satisfies the following properties:

Correctness: *For every* m *in the commitment message space* M_{ck}*, if* $(\mathsf{ck} \leftarrow \mathsf{Setup}(1^\kappa)$*,* $(\mathsf{com}, o) \leftarrow \mathsf{Commit}_{\mathsf{ck}}(m)$*, then* $\mathsf{Open}_{\mathsf{ck}}(\mathsf{com}, m, o) = 1$*.*

Hiding: *It is computationally infeasible for any adversary* $\mathcal{A}$ *to distinguish between commitments* $\mathsf{com}_0, \mathsf{com}_1$ *to messages* $m_0, m_1 \in M_{\mathsf{ck}}$ *(where* $(\mathsf{com}_b, o_b) \leftarrow \mathsf{Commit}_{\mathsf{ck}}(m_b)$*).*

Binding: *It is computationally infeasible for any adversary* $\mathcal{A}$ *to find values* $(\mathsf{com}, m_0, m_1, o_0, o_1)$ *such that* com *can be opened to both* (m_0, o_0) *and* (m_1, o_1)*.*

Formally, for every PPT adversary $\mathcal{A}$*, the following probability is negligible:*

$$\Pr\left[\begin{array}{c} m_0 \neq m_1 \\ \wedge\ \mathsf{Open}_{\mathsf{ck}}(\mathsf{com}, m_0, o_0) = 1 \\ \wedge\ \mathsf{Open}_{\mathsf{ck}}(\mathsf{com}, m_1, o_1) = 1 \end{array} \middle| \begin{array}{c} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\kappa) \\ (\mathsf{com}, m_0, m_1, o_0, o_1) \leftarrow \mathcal{A}(1^\kappa, \mathsf{ck}) \end{array}\right]$$

Definition 7.13
Our generic construction will be the $\mathcal{VC}$ *scheme from Definition 7.6 instantiated with a circuit-private FHE scheme, but using a SNARK for the following relation* $\mathcal{R}_F$*:*

$$\mathcal{R}_F = \left\{ \left((\mathbf{ct}_{out}, \mathsf{Com}_{in}), (C_{in}, O_{in}, C'_{in})\right) \middle| \begin{array}{c} \mathbf{ct}_{out} = \mathsf{Eval}_{\mathsf{ek}}(F, C_{in}, C'_{in}) \\ \wedge\ \forall i : C_{in}[i] \in \Omega \\ \wedge\ \forall i : C'_{in}[i] \in \Omega \\ \wedge\ \forall i : \mathsf{Open}_{\mathsf{ck}}(\mathsf{Com}_{in}[i], C_{in}[i], O_{in}[i]) \end{array} \right\}$$

The algorithm for $\mathcal{VC}$ *need to be adapted as follows:*

$\mathsf{Setup}(1^\kappa):$ Setup *additionally generates the commit key* ck*, which is independent of the circuit to be evaluated;*

$\mathsf{KGen}_{\mathsf{pk}}(f):$ *does not change (insofar as it only generates keying material for another SNARK);*

$\mathsf{ProbGen}_{\mathsf{pk}}(x):$ $\sigma_x = (C_{in}, O_{in})$ *remains unchanged, but* $\tau_x = \mathsf{Com}_{in}$ *now consists of commitment and opening values for inputs in* x*;*

$\mathsf{Compute}_{\mathsf{pk}_f}(\sigma_x, w)$: *remains unchanged (but uses a SNARK for $\mathcal{R}_F$ as defined above);*

$\mathsf{Verify}_{\mathsf{sk}_f}(\tau_x, \sigma_y)$: *the verification step simply uses the SNARK to check if the proof included in σ_y is correct;*

$\mathsf{Decode}_{\mathsf{sk},\mathsf{sk}_f}(\sigma_y)$: *remains unchanged.*

Theorem 7.14
Let $\mathcal{VC}$ be the generic VC scheme from Definition 7.13. Then $\mathcal{VC}$ is a maliciously-secure VC with correctness, input hiding, security, and context-hiding.

Proof Correctness follows directly from Proof 4. Security follows from Proof 6, due to the binding property of the commitment scheme. Input privacy can also be reduced to the input privacy of the server context-hiding scheme, by noting that the adversary does not gain advantage by the presence of the commitment values in τ_x, owing to the hiding property of the commitment.

In order to prove that $\mathcal{VC}$ is context hiding, we simply define the simulators S_1, S_2, S_3, and S_τ from Definition 5.4:

$S_1(1^\kappa) \to (\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{td})$: S_1 simply runs $\mathsf{Setup}(1^\kappa)$, and outputs an empty trapdoor td.

$S_2(\mathsf{td}, f) \to (\mathsf{pk}_f^*, \mathsf{sk}_f^*, \mathsf{td}_f)$: S_2 proceeds as $\mathsf{KGen}_{\mathsf{pk}}(f)$, but uses $(\mathsf{crs}_f, \mathsf{vk}_f, \mathsf{td}_f) \leftarrow \Pi.\mathsf{SetupSim}(1^\kappa)$ instead of the real values.

$S_3(\mathsf{td}_f, \tau_x, f(x, w) \to \sigma_y^*$: s_3 first computes a fake result ciphertext $\mathbf{ct}_{\text{out}}^* \leftarrow \mathcal{E}.\mathsf{Sim}(\mathsf{pk}, f(x, w))$ by using $\mathcal{E}$'s circuit privacy simulator, and then generates a simulated proof π^* for the statement $(\mathbf{ct}_{\text{out}}^*, \mathsf{Com}_{\text{in}})$ using the SNARK simulator and the trapdoor td_f.

$S_\tau(\mathsf{td}) \to \tau_x$: S_τ chooses a dummy value, and commits to it.

S_1 obviously generates the same distribution as Setup. S_2 also generates a distribution that is indistinguishable from KGen due to the zero-knowledge property of the SNARK. S_3 generates $\sigma_y^* = (\mathbf{ct}_{\text{out}}^*, \pi^*)$ indistinguishable from σ_y, owing to the circuit privacy of $\mathcal{E}$ and the zero-knowledge property of the SNARK. S_τ generates commitments to dummy values that are indistinguishable from commitments to x due to the commitment scheme's hiding property. □

7.2 Input Checks

An input check can either be implemented in the circuit directly if it is easily expressible as a polynomial; in this case, correctness with input checks directly follows from the correctness of the underlying FHE scheme.

As noted in [Boi+21], more complex checks can also be implemented by embedding them inside the relation $\mathcal{R}_f$ (i.e., instead of proving $\mathbf{ct} \in \Omega$, one would prove $\mathbf{ct} \in \Omega \wedge \mathsf{check}(\mathbf{ct}) = 1$, for example by relying on an auxiliary SNARK). In this case, correctness with checks follows from the correctness of the SNARK. One could even imagine a combined approach for complex checks, where part of the check (e.g., simple linear equations) are implemented directly in the circuit, and the remaining part (e.g., non-linear requirements) is checked using the SNARK.

7.3 Input Commitment

We define similar relations than in Definition 7.4, except that we require the server's value to be committed, and that this commitment is proved to be opened:

Let us now define the two following relations and corresponding O-SNARKs:

Definition 7.15 (FHE Integrity Relation with Commitments)
Let $\mathcal{R}_{\mathsf{Enc}}$ from Definition 7.4 capture valid encryptions, and let $\mathcal{R}_{F,\mathsf{com}}$ capture correct evaluations of a function F:

$$\mathcal{R}_{F,\mathsf{com}} = \left\{ \begin{array}{l|c} \big((\mathbf{ct}_{out}, \mathsf{Com}_{in}), & \mathbf{ct}_{out} = \mathsf{Eval}_{\mathsf{ek}}(F, C_{in}, C'_{in}) \\ & \wedge \forall i : C_{in}[i] \in \Omega \\ (C_{in}, O_{in}, C'_{in})\big) & \wedge \forall i : C'_{in}[i] \in \Omega \\ & \wedge \forall i : \mathsf{Open}_{\mathsf{ck}}(\mathsf{Com}_{in}[i], C'_{in}[i], O_{in}[i]) \end{array} \right\}$$

This is exactly the relation from Definition 7.13, except that the commitment is now to the server input instead of the client input. In the following, let $\Pi_{F,\mathsf{com}}$ be an O-SNARK for $\mathcal{R}_{F,\mathsf{com}}$.

Definition 7.16 (Generic Construction for Input Commitment)
For an FHE scheme $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ and the zk-O-SNARKs from Definition 7.4, we define a new FHE scheme $\mathcal{VC}[\mathcal{E}]$ as follows. $\mathcal{VC}[\mathcal{E}]$ is the same as in Definition 7.6, except that the relation $\mathcal{R}_F$ from Definition 7.15 is used instead of the one from Definition 7.4. Correspondingly, both the verification and the proof are done using the statement $(\mathbf{ct}_{out}, \mathsf{Com}_{in})$ instead of simply $(\mathbf{ct}_{out}, C_{in})$, where Com_{in} and $O_{in}$are sent by the serve to the client at the start of the protocol.

Theorem 7.17
Let $\mathcal{E}$ be an $\mathcal{F}$-IND-CPA$^{\mathsf{D}}$-secure FHE scheme, and let $\mathcal{VC}[\mathcal{E}]$ be the scheme from Definition 7.16. Then $\mathcal{VC}[\mathcal{E}]$ is correct, PRIV-CCA1*-secure,* VER-CCA1-*secure, and offers input commitment.*

Proof

Correctness. As in Definition 5.1, correctness follows immediately from the correctness of the FHE and proof schemes.

Input Privacy. The proof is very similar to Proof 5, as none of the changes made to the protocol (the commitments Com_{in}, the proving and verification procedure) are dependent on x.

Security. The proof follows immediately from Proof 6.

Input Commitment. We can reduce input commitment to the binding property of the commitment scheme. Given any PPT adversary $\mathcal{A}$ against COM, we can construct an adversary $\mathcal{B}$ for the binding security game of the commitment scheme underlying $\mathcal{VC}$. $\mathcal{B}$ can perfectly simulate the COM game for $\mathcal{A}$, and will receive x, $w = M_{\text{in}}$, and $w' = M'_{\text{in}}$ such that for honestly computed σ_y and σ'_y from x and w (respectively w') are both accepted by an honest verifier, and $w \neq w'$. $\mathcal{B}$ can select the index i such that $M_{\text{in}}[i] \neq M'_{\text{in}}[i]$, and output $(\mathsf{Com}_{\text{in}}[i], M'_{\text{in}}[i], O_{\text{in}}[i], M_{\text{in}}[i], O_{\text{in}}[i])$. $\mathcal{B}$ wins if and only if wins. We note here that we achieve input commitment (a property unclaimed by [Boi+21]) with a weaker assumption on the commitment scheme (namely, binding instead of knowledge binding, cf. [Boi+21, Definition 10]). $\square$

Chapter 8

Towards Better Primitives for Robust FHE Integrity Protection

In this final chapter, we evaluate how suitable the primitives investigated in Chapter 4 are for our generic constructions from Chapter 7, and sketch avenues of research towards integrity primitives that support the complex requirements of practical FHE deployments.

8.1 Cryptography-Based

Our generic construction requires O-SNARKs to prove correct encryption and correct evaluation. These SNARKs must be robust in the face of oracles (in particular, verification and decryption oracles). Additionally, they should support commitments and advanced input checking in order to support Input Commitment and Input Checks.

A given protocol that is insecure in the face of verification queries can still be used securely in practice by aborting the protocol and generating new keys after a failed verification. This would strip a malicious adversary from a decryption oracle (for the keying material used up to the abort).

Efficient Input Checks. Input checks must be expressible as ring operations in order to be natively checked using ring-based proofs. For example, consider a *zero slots check:* this check would verify that for a plaintext input $\mathbf{m} \in R_t \cong \mathbb{Z}_t^N$, the slot values at indices S_0 of $\mathbf{m}$ are zero, i.e., $\forall i \in S_0 : \mathbf{m}[i] = 0$. This can be enforced easily by first embedding the plaintext in cipherspace (e.g., by setting the second ciphertext component to 0), and then performing a multiplication with a constant mask $\mathbf{m}_{\text{mask}}$, where $\mathbf{m}_{\text{mask}}[i] = 0$ if $i \in S_0$, and 1 otherwise.

Similarly, *range checks* can be implemented by taking advantage of the RNS representation as follows. for a given plaintext $\mathbf{m}$, we want to check that each coefficient $\mathbf{m}_i$ lies in $\mathbb{Z}_k$, for $k \leq t$. This can be done by embedding $\mathbf{m}$ into R_q, multiplying each element by $\delta := \lceil \frac{q_1}{k} \rceil$, applying a modulus switch to q_1, and scaling back (with rounding) to R_t. Any element $\mathbf{m}_i \in \mathbb{Z}_k$ will be preserved by this transformation, while $\mathbf{m}_i \in \mathbb{Z}_q \setminus \mathbb{Z}_k$ will be mapped to an element of $\mathbb{Z}_k$. We note that this process can be used to enforce more fine-grained constraints $\mathbf{m}_i \in [a, b]$ by enforcing $\tilde{\mathbf{m}}_i := \mathbf{m}_i - a \in \mathbb{Z}_b$; b has to be the same for the entire plaintext, but the lower bound a can be chosen for each coefficient independently.

Efficient Input Commitments. Ring-based proof systems can support commitments if the opening algorithm can be expressed as ring operations. This could be done by using a lattice-based commitment scheme (e.g., the BDOP commitment [Bau+18]); in [CTPH21], Chatel et al. show how a BDOP commitment can be opened homomorphically using CKKS, a technique which could be adapted to ring-based proofs. We leave concrete details for future work.

8.2 Hardware-Based

Hardware-based primitives (i.e., TEEs) are much more flexible integrity primitives than their cryptographic counterparts; in particular, they implement proof systems that are secure against verification oracles.

Input checks can be implemented straightforwardly for any check expressible as a program. Similarly, input commitment can be trivially implemented by relying on any classical commitment scheme (e.g., over bits, or even lattice-based), adding a minimal overhead compared to the FHE computation.

Proof of Computation. One big advantage of TEEs is that they not only allow for proofs of knowledge, but also for proofs that a computation actually took place. This unique feature allows to natively producing proofs-of-computation: for this, we require that an adversary is not able to expend less computational power in order to provide an output for a given computation. This can be enforced by using a "no-shortcut" FHE library, i.e., a standard FHE library modified such that each FHE operations requires the same amount of computation, regardless of the operation's inputs. In practice, this would only mean to remove support short-circuit evaluation for ciphertext-plaintext operations (e.g., optimizations that make a multiplication output 0 if one of the (plaintext) operands is 0), as most FHE operations are oblivious to the contents of its inputs.

We thus see promising avenues of research for both hardware-based and cryptographic integrity primitives. In the short term, hardware primitives can easily be deployed at minimal cost in order to quickly mitigate the attacks we have outlined in Chapter 5 (e.g., by using our framework [Knaa]). However, TEEs require some trust in the hardware vendor, and might be not readily available for all use cases. Therefore, we believe that more practical cryptographic integrity primitives are an interesting and promising avenue of research (and as we have shown in Chapter 4, cryptographic and hardware-based approaches can complement each other for further efficiency and expressiveness).

Chapter 9

Conclusion and Future Work

FHE is a very promising privacy-preserving technology on the brink of deployment, but due to its lack of integrity, using it in real-world settings can lead to a total loss of correctness and privacy for users. In this thesis, we first studied, analysed and compared existing approaches and notions for FHE integrity in the literature, and outlined limitations of these approaches and their inadequacy for state-of-the art FHE schemes. We followed up this analysis by extracting desirable security and functional properties for major types of FHE applications, and showed how existing integrity notions fail to guarantee these properties. We then formalised these desirable properties as novel, more robust integrity notions that naturally generalise existing definitions. We also studied more traditional, indistinguishability notions for FHE, and provided new insights into their connection to verifiable computation.

In order to address this mismatch between theory and practice, we then provide generic construction that achieve our newly defined integrity notions, and prove their security. Finally, we presented desirable criteria for practical integrity primitives, and we analysed, improved, and implemented the two most promising approaches (both cryptographic and hardware-based). As a last step, we sketched ideas for more concretely efficient integrity primitives that fully support the needs of modern FHE.

This constitutes a large avenue of future work, with the aim to construct more efficient, flexible, and robust integrity primitives. Another avenue of research lies in further strengthening and refining the adversarial model for FHE applications in practice, in particular for many-party systems. Finally, recent years have seen a variety of advanced FHE variants, such as functional bootstrapping, transciphering between FHE schemes, and even transciphering between a traditional FHE scheme and a custom stream-cipher. We consider that investigating the security of these advanced variants, and to extend our constructions and analysis to these settings is an interesting venue for future work.

Bibliography

[AB+22] Ahmad Al Badawi et al. “OpenFHE: Open-Source Fully Homomorphic Encryption Library”. In: *Cryptology ePrint Archive* (2022).

[ABS15] Alberto Torres Wilson Abel, Nandita Bhattacharjee, and Bala Srinivasan. “Privacy-preserving biometrics authentication systems using fully homomorphic encryption”. In: *Int. J. Pervasive Comput. Commun.* 11.2 (Jan. 2015), pp. 151–168.

[AEH15] Louis J M Aslett, Pedro M Esperança, and Chris C Holmes. “A review of homomorphic encryption and software tools for encrypted statistical machine learning”. In: (Aug. 2015). arXiv: `1508.06574 [stat.ML]`.

[Ali+21] Asra Ali et al. “{Communication–Computation} Trade-offs in {PIR}”. In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 1811–1828.

[Amd] *AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More*. `https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf`.

[Ang+18] Sebastian Angel et al. “PIR with Compressed Queries and Amortized Query Processing”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. May 2018, pp. 962–979.

[Bau+18] Carsten Baum et al. “More Efficient Commitments from Structured Lattice Assumptions”. In: *Security and Cryptography for Networks*. Springer International Publishing, 2018, pp. 368–385.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. “(Leveled) Fully Homomorphic Encryption without Bootstrapping”. In: *ACM Trans. Comput. Theory* 6.3 (July 2014), pp. 1–36.

[Bis+18] Anurag Bishnoi et al. “On Zeros of a Polynomial in a Finite Grid”. In: *Comb. Probab. Comput.* 27.3 (May 2018), pp. 310–333.

[BMA20] Adrian Brasoveanu, Megan Moodie, and Rakshit Agrawal. “GPS: Integration of Graphene, PALISADE, and SGX forLarge-scale Aggregations of Distributed Data”. In: *CEUR Workshop Proceedings*. Vol. 2657. CEUR-WS, 2020, pp. 1–9.

[Boe+21] Fabian Boemer et al. *Intel HEXL (release 1.2)*. `https://github.com/intel/hexl`. 2021.

[Boi+21] Alexandre Bois et al. “Flexible and Efficient Verifiable Computation on Encrypted Data”. In: *Public-Key Cryptography – PKC 2021*. Springer International Publishing, 2021, pp. 528–558.

[Bon+07] Dan Boneh et al. “Chosen-Ciphertext Security from Identity-Based Encryption”. In: *SIAM J. Comput.* 36.5 (Jan. 2007), pp. 1301–1328.

[Bre+22] Lars Brenna et al. “TFHE-rs: A library for safe and secure remote computing using fully homomorphic encryption and trusted execution environments”. In: *Array* 13 (Mar. 2022), p. 100118.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. “Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages”. In: *Advances in Cryptology – CRYPTO 2011*. Springer Berlin Heidelberg, 2011, pp. 505–524.

[Can+17] Ran Canetti et al. “Chosen-Ciphertext Secure Fully Homomorphic Encryption”. In: *Public-Key Cryptography – PKC 2017*. Springer Berlin Heidelberg, 2017, pp. 213–240.

[CF13] Dario Catalano and Dario Fiore. “Practical Homomorphic MACs for Arithmetic Circuits”. In: *Advances in Cryptology – EUROCRYPT 2013*. Springer Berlin Heidelberg, 2013, pp. 336–352.

[CGG16] Ilaria Chillotti, Nicolas Gama, and Louis Goubin. “Attacking FHE-based applications by software fault injections”. In: *Cryptology ePrint Archive* (2016).

[Cha+22] Sylvain Chatel et al. “Verifiable Encodings for Secure Homomorphic Analytics”. In: (July 2022). arXiv: `2207.14071 [cs.CR]`.

[Che+17] Jung Hee Cheon et al. “Homomorphic Encryption for Arithmetic of Approximate Numbers”. In: *Advances in Cryptology – ASIACRYPT 2017*. Springer International Publishing, 2017, pp. 409–437.

[Che+18] Hao Chen et al. “Labeled PSI from Fully Homomorphic Encryption with Malicious Security”. In: *Cryptology ePrint Archive* (2018).

[Chi+20] Ilaria Chillotti et al. "TFHE: Fast Fully Homomorphic Encryption Over the Torus". In: *J. Cryptology* 33.1 (Jan. 2020), pp. 34–91.

[CLR17] Hao Chen, Kim Laine, and Peter Rindal. "Fast Private Set Intersection from Homomorphic Encryption". In: *Cryptology ePrint Archive* (2017).

[Con+21] Kelong Cong et al. "Labeled PSI from Homomorphic Encryption with Reduced Computation and Communication". In: *Cryptology ePrint Archive* (2021).

[Con+22] Kelong Cong et al. "SortingHat: Efficient Private Decision Tree Evaluation via Homomorphic Encryption and Transciphering". In: *Cryptology ePrint Archive* (2022).

[Cop+21] Luigi Coppolino et al. "VISE: Combining Intel SGX and Homomorphic Encryption for Cloud Industrial Control Systems". In: *IEEE Trans. Comput.* 70.5 (May 2021), pp. 711–724.

[CT15] Massimo Chenal and Qiang Tang. "On Key Recovery Attacks Against Existing Somewhat Homomorphic Encryption Schemes". In: *Progress in Cryptology - LATINCRYPT 2014*. Springer International Publishing, 2015, pp. 239–258.

[CTPH21] Sylvain Chatel, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. "Privacy and integrity preserving computations with {CRISP}". In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 2111–2128.

[DDA13] Angsuman Das, Sabyasachi Dutta, and Avishek Adhikari. "Indistinguishability against Chosen Ciphertext Verification Attack Revisited: The Complete Picture". In: *Provable Security*. Springer Berlin Heidelberg, 2013, pp. 104–120.

[DG17] Nir Drucker and Shay Gueron. "Combining Homomorphic Encryption with Trusted Execution Environment: A Demonstration with Paillier Encryption and SGX". In: *Proceedings of the 2017 International Workshop on Managing Insider Security Threats*. Vol. 2017-January. MIST '17. Dallas, Texas, USA: Association for Computing Machinery, Oct. 2017, pp. 85–88.

[DM15] Léo Ducas and Daniele Micciancio. "FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second". In: *Advances in Cryptology – EUROCRYPT 2015*. Springer Berlin Heidelberg, 2015, pp. 617–640.

[Emu+18] Keita Emura et al. "Chosen ciphertext secure keyed-homomorphic public-key cryptosystems". en. In: *Des. Codes Cryptogr.* 86.8 (Aug. 2018), pp. 1623–1683.

[Emu21] Keita Emura. “On the Security of Keyed-Homomorphic PKE: Preventing Key Recovery Attacks and Ciphertext Validity Attacks”. In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E104.A.1 (2021), pp. 310–314.

[FGP14] Dario Fiore, Rosario Gennaro, and Valerio Pastro. “Efficiently Verifiable Computation on Encrypted Data”. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security.* CCS ’14. Scottsdale, Arizona, USA: Association for Computing Machinery, Nov. 2014, pp. 844–855.

[FHR21] Prastudy Fauzi, Martha Norberg Hovd, and Håvard Raddum. “On the IND-CCA1 Security of FHE Schemes”. In: *Cryptology ePrint Archive* (2021).

[FN16] Dario Fiore and Anca Nitulescu. “On the (in)security of SNARKs in the presence of oracles”. In: *Theory of Cryptography.* Lecture notes in computer science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 108–138.

[FNP20] Dario Fiore, Anca Nitulescu, and David Pointcheval. “Boosting Verifiable Computation on Encrypted Data”. In: *Public-Key Cryptography – PKC 2020.* Springer International Publishing, 2020, pp. 124–154.

[FV12] Junfeng Fan and Frederik Vercauteren. “Somewhat Practical Fully Homomorphic Encryption”. In: *Cryptology ePrint Archive* (2012).

[GB+16] Ran Gilad-Bachrach et al. “CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy”. In: *Proceedings of The 33rd International Conference on Machine Learning.* Ed. by Maria Florina Balcan and Kilian Q Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 2016, pp. 201–210.

[Gea15] Jon Geater. “ARM® TrustZone®”. In: *Trusted Computing for Embedded Systems.* Ed. by Bernard Candaele, Dimitrios Soudris, and Iraklis Anagnostopoulos. Cham: Springer International Publishing, 2015, pp. 35–45.

[Gen09] Craig Gentry. “Fully homomorphic encryption using ideal lattices”. In: *Proceedings of the forty-first annual ACM symposium on Theory of computing.* STOC ’09. Bethesda, MD, USA: Association for Computing Machinery, May 2009, pp. 169–178.

[Geu18] Martijn Geurden. “A New Future for Military Security Using Fully Homomorphic Encryption”. PhD thesis. 2018.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. "Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers". In: *Advances in Cryptology – CRYPTO 2010*. Springer Berlin Heidelberg, 2010, pp. 465–482.

[GH19] Craig Gentry and Shai Halevi. "Compressible FHE with Applications to PIR". In: *Theory of Cryptography*. Springer International Publishing, 2019, pp. 438–464.

[Gje+17] Anders T Gjerdrum et al. "Performance of trusted computing in cloud infrastructures with Intel SGX". In: *Proceedings of the 7th International Conference on Cloud Computing and Services Science*. Porto, Portugal: SCITEPRESS - Science and Technology Publications, 2017, pp. 668–675.

[GNSV21] Chaya Ganesh, Anca Nitulescu, and Eduardo Soria-Vazquez. "Rinocchio: SNARKs for Ring Arithmetic". In: *Cryptology ePrint Archive* (2021).

[Gol+13] Shafi Goldwasser et al. "How to Run Turing Machines on Encrypted Data". In: *Advances in Cryptology – CRYPTO 2013*. Springer Berlin Heidelberg, 2013, pp. 536–553.

[GW13] Rosario Gennaro and Daniel Wichs. "Fully Homomorphic Message Authenticators". In: *Advances in Cryptology - ASIACRYPT 2013*. Springer Berlin Heidelberg, 2013, pp. 301–320.

[Iez20] Michela Iezzi. "Practical Privacy-Preserving Data Science With Homomorphic Encryption: An Overview". In: *2020 IEEE International Conference on Big Data (Big Data)*. Dec. 2020, pp. 3979–3988.

[Iso] *ISO/IEC AWI 18033-8*. en. `https://www.iso.org/standard/83139.html`. Accessed: 2022-10-7. 2021.

[Kim+18] Miran Kim et al. "Secure Logistic Regression Based on Homomorphic Encryption: Design and Evaluation". en. In: *JMIR Med Inform* 6.2 (Apr. 2018), e19.

[Knaa] Christian Knabenhans. *FHE-in-TEE: a framework to run Fully Homomorphic Encryption applications on Trusted Execution Environments, with support for untrusted hardware acceleration.* `https://github.com/MarbleHE/fhe-in-tee`.

[Knab] Christian Knabenhans. *rinocchio: a generic implementation of the Rinocchio protocol, with SEAL bindings.* `https://github.com/MarbleHE/rinocchio-SEAL`.

[KO97] E Kushilevitz and R Ostrovsky. "Replication is not needed: single database, computationally-private information retrieval". In: *Proceedings 38th Annual Symposium on Foundations of Computer Science*. Oct. 1997, pp. 364–373.

[KPZ21] Andrey Kim, Yuriy Polyakov, and Vincent Zucca. "Revisiting Homomorphic Encryption Schemes for Finite Fields". In: *Cryptology ePrint Archive* (2021).

[Lai+16] Junzuo Lai et al. "CCA-Secure Keyed-Fully Homomorphic Encryption". In: *Public-Key Cryptography – PKC 2016*. Springer Berlin Heidelberg, 2016, pp. 70–98.

[Lau+21] Kristin Lauter et al. *Password Monitor: Safeguarding passwords in Microsoft Edge*. en. 2021.

[Lee+22] Joon-Woo Lee et al. "Privacy-Preserving Machine Learning With Fully Homomorphic Encryption for Deep Neural Network". In: *IEEE Access* 10 (2022), pp. 30039–30054.

[LGM16] Zengpeng Li, Steven D Galbraith, and Chunguang Ma. "Preventing Adaptive Key Recovery Attacks on the GSW Levelled Homomorphic Encryption Scheme". In: *Provable Security*. Springer International Publishing, 2016, pp. 373–383.

[Li+22] Baiyu Li et al. "Securing Approximate Homomorphic Encryption Using Differential Privacy". In: *Cryptology ePrint Archive* (2022).

[LM21] Baiyu Li and Daniele Micciancio. "On the Security of Homomorphic Encryption on Approximate Numbers". In: *Advances in Cryptology – EUROCRYPT 2021*. Springer International Publishing, 2021, pp. 648–677.

[Lof+12] Jake Loftus et al. "On CCA-Secure Somewhat Homomorphic Encryption". In: *Selected Areas in Cryptography*. Springer Berlin Heidelberg, 2012, pp. 55–72.

[LWZ18] Shimin Li, Xin Wang, and Rui Zhang. "Privacy-Preserving Homomorphic MACs with Efficient Verification". In: *Web Services – ICWS 2018*. Springer International Publishing, 2018, pp. 100–115.

[MBo20] Mouchet, Bossuat, and others. "Lattigo: A multiparty homomorphic encryption library in go". In: *WAHC 2020–8th* (2020).

[MCR21] Muhammad Haris Mughees, Hao Chen, and Ling Ren. "OnionPIR: Response Efficient Single-Server PIR". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. CCS '21. Virtual Event, Republic of Korea: Association for Computing Machinery, Nov. 2021, pp. 2292–2306.

[Mur+20] Kit Murdock et al. "Plundervolt: Software-based Fault Injection Attacks against Intel SGX". In: *2020 IEEE Symposium on Security and Privacy (SP)*. San Francisco, CA, USA: IEEE, May 2020, pp. 1466–1482.

[Nat+21] Deepika Natarajan et al. “CHEX-MIX: Combining Homomorphic Encryption with Trusted Execution Environments for Two-party Oblivious Inference in the Cloud”. In: *Cryptology ePrint Archive* (2021).

[NBB20] Alexander Nilsson, Pegah Nikbakht Bideh, and Joakim Brorsson. “A Survey of Published Attacks on Intel SGX”. In: (June 2020). arXiv: `2006.13598 [cs.CR]`.

[NY90] M Naor and M Yung. “Public-key cryptosystems provably secure against chosen ciphertext attacks”. In: *Proceedings of the twenty-second annual ACM symposium on Theory of Computing*. STOC ’90. Baltimore, Maryland, USA: Association for Computing Machinery, Apr. 1990, pp. 427–437.

[Oe] *Open Enclave SDK*. `https://github.com/openenclave/openenclave`.

[Oze+21] Ozgun Ozerk et al. “Efficient Number Theoretic Transform Implementation on GPU for Homomorphic Encryption”. In: *Cryptology ePrint Archive* (2021).

[PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. “How to Delegate and Verify in Public: Verifiable Computation from Attribute-Based Encryption”. In: *Theory of Cryptography*. Springer Berlin Heidelberg, 2012, pp. 422–439.

[PT20] Jeongeun Park and Mehdi Tibouchi. “SHECS-PIR: Somewhat Homomorphic Encryption-Based Compact and Scalable Private Information Retrieval”. In: *Computer Security – ESORICS 2020*. Springer International Publishing, 2020, pp. 86–106.

[Sad+19] Md Nazmus Sadat et al. “SAFETY: Secure gwAs in Federated Environment through a hYbrid Solution”. en. In: *IEEE/ACM Trans. Comput. Biol. Bioinform.* 16.1 (Jan. 2019), pp. 93–102.

[SAK17] Naw Safrin Sattar, Muhammad Abdullah Adnan, and Maimuna Begum Kali. “Secured aerial photography using Homomorphic Encryption”. In: *2017 International Conference on Networking, Systems and Security (NSysS)*. ieeexplore.ieee.org, Jan. 2017, pp. 107–114.

[Sea] *Microsoft SEAL (release 4.0)*. `https://github.com/Microsoft/SEAL`. Mar. 2022.

[SET22] Shingo Sato, Keita Emura, and Atsushi Takayasu. “Keyed-Fully Homomorphic Encryption without Indistinguishability Obfuscation”. In: *Cryptology ePrint Archive* (2022).

[Sgx] *Intel® Software Guard Extensions (Intel® SGX)*. en. https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions.html. Accessed: 2022-10-7.

[Sun+20] Xiaoqiang Sun et al. "A Survey on Secure Computation Based on Homomorphic Encryption in Vehicular Ad Hoc Networks". en. In: *Sensors* 20.15 (July 2020).

[TB18] Florian Tramèr and Dan Boneh. "Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware". In: (June 2018). arXiv: 1806.03287 [stat.ML].

[TBS14] Wilson Abel Alberto Torres, Nandita Bhattacharjee, and Bala Srinivasan. "Effectiveness of Fully Homomorphic Encryption to Preserve the Privacy of Biometric Data". In: *Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services*. iiWAS '14. Hanoi, Viet Nam: Association for Computing Machinery, Dec. 2014, pp. 152–158.

[Tra+17] Florian Tramèr et al. "Sealed-Glass Proofs: Using Transparent Enclaves to Prove and Sell Knowledge". In: *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. Apr. 2017, pp. 19–34.

[Tu+22] Binbin Tu et al. "Fast Unbalanced Private Set Union from Fully Homomorphic Encryption". In: *Cryptology ePrint Archive* (2022).

[VJH21] Alexander Viand, Patrick Jattke, and Anwar Hithnawi. "SoK: Fully Homomorphic Encryption Compilers". In: *2021 IEEE Symposium on Security and Privacy (SP)*. ieeexplore.ieee.org, May 2021, pp. 1092–1108.

[VK] Alexander Viand and Christian Knabenhans. *polytools: utilities for Microsoft SEAL's polynomial arithmetic*.

[Wan+18] Xiaofeng Wang et al. "iDASH secure genome analysis competition 2017". en. In: *BMC Med. Genomics* 11.Suppl 4 (Oct. 2018), p. 85.

[Wan+19] Wenhao Wang et al. "Toward Scalable Fully Homomorphic Encryption Through Light Trusted Computing Assistance". In: (May 2019). arXiv: 1905.07766 [cs.CR].

[WNK20] Alexander Wood, Kayvan Najarian, and Delaram Kahrobaei. "Homomorphic Encryption for Machine Learning in Medicine and Bioinformatics". In: *ACM Comput. Surv.* 53.4 (Aug. 2020), pp. 1–35.

[WWX18] Biao Wang, Xueqing Wang, and Rui Xue. "CCA1 secure FHE from PIO, revisited". en. In: *Cybersecurity* 1.1 (Sept. 2018), pp. 1–8.

[Yao82] Andrew C Yao. “Protocols for secure computations”. In: *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*. Nov. 1982, pp. 160–164.

[Yao86] Andrew Chi-Chih Yao. “How to generate and exchange secrets”. In: *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*. Oct. 1986, pp. 162–167.

[ZPS12a] Zhenfei Zhang, Thomas Plantard, and Willy Susilo. *On the CCA-1 Security of Somewhat Homomorphic Encryption over the Integers*. 2012.

[ZPS12b] Zhenfei Zhang, Thomas Plantard, and Willy Susilo. “Reaction Attack on Outsourced Computing with Fully Homomorphic Encryption Schemes”. In: *Information Security and Cryptology - ICISC 2011*. Springer Berlin Heidelberg, 2012, pp. 419–436.