vFHE: Verifiable Fully Homomorphic Encryption

Christian Knabenhans
EPFL
Lausanne, Switzerland
christian.knabenhans@epfl.ch

Alexander Viand
Intel Labs
Zurich, Switzerland
alexander.viand@intel.com

Antonio Merino-Gallardo
ETH Zurich
Zurich, Switzerland
antonio@m-g.es

Anwar Hithnawi
University of Toronto
Toronto, Canada
anwar.hithnawi@cs.toronto.edu

Abstract

Fully Homomorphic Encryption (FHE) is a powerful building block for secure and private applications. However, state-of-the-art FHE schemes do not offer any integrity guarantees, which can lead to devastating correctness and security issues when FHE is deployed in non-trivial settings. In this paper, we take a critical look at existing integrity solutions for FHE, and analyze their (often implicit) threat models, efficiency, and adequacy with real-world FHE deployments. We explore challenges of what we believe is the most flexible and promising integrity solution for FHE: namely, zero-knowledge Succinct Non-interactive ARguments of Knowledge (zkSNARKs); we showcase optimizations for both general-purpose zkSNARKs and zkSNARKs designed for FHE. We then present two software frameworks, circomlib-FHE and zkOpenFHE, which allow practitioners to automatically augment existing FHE pipelines with integrity guarantees. Finally, we leverage our tools to evaluate and compare different approaches to FHE integrity, and discuss open problems that stand in the way of a widespread deployment of FHE in real-world applications.

CCS Concepts

• **Security and privacy** → **Cryptography**; **Management and querying of encrypted data**.

Keywords

FHE Integrity; Zero-Knowledge Proofs; OpenFHE

ACM Reference Format:
Christian Knabenhans, Alexander Viand, Antonio Merino-Gallardo, and Anwar Hithnawi. 2024. vFHE: Verifiable Fully Homomorphic Encryption. In *Proceedings of the 12th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3689945.3694806

WAHC '24, October 14–18, 2024, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-1241-8/24/10
https://doi.org/10.1145/3689945.3694806

1 Introduction

In recent years, there has been an increased interest in applying secure computation techniques to both strengthen security and privacy guarantees of existing applications, and to unlock the benefits of previously inaccessible data. In the last few years, Fully Homomorphic Encryption (FHE) has undergone several leaps in performance that make it a compelling solution for a variety of secure computation applications. Cryptographic and algorithmic improvements in the last decade have given us mature and efficient schemes, and an improved understanding of how to apply them has led to a significant reduction of the overhead FHE introduces [70]. For example, in the span of two years, the community improved the runtime for encrypted inference of simple models from 300 seconds to 0.03 seconds [39, 43, 61]. An improved understanding of how to exploit existing hardware (e.g., GPUs, TPUs, and FPGAs) and the development of custom FHE accelerators [9, 24, 35, 65–67] promises to reduce wall-clock times even further. As a result, there has been a shift from demonstrating proof-of-concept applications to focusing on solving real-world problems [42, 73] and we have seen a first wave of real-world deployments of FHE-based applications emerge. For example, Microsoft's Edge browser includes an FHE-based password monitoring system [50], the Korean government piloted an FHE-based contact tracing application [1], and Apple recently introduced FHE-based Live Caller ID Lookup for iOS [2, 3].

As FHE matures and interest in deploying it in production increases, challenges in development and deployment are now rising to the forefront. There have been significant efforts dedicated to lowering the barrier of entry to FHE development [40, 69, 74], putting the development of efficient FHE applications within reach of a much larger audience. These works, however, have so far mostly ignored the fundamental challenges that arise when trying to realize secure-computation–based applications in practice: cryptographic considerations now need to be interleaved with the application, rather than occurring at a clearly defined edge. Naturally, this requires augmenting policies and infrastructures for, e.g., key distribution and management to support these new application flows. However, it also raises more fundamental issues, as we need to contend with the challenges of deploying FHE in complex real-world settings that frequently violate the basic assumptions underlying most proposed FHE applications.

The vast majority of existing works on FHE ignore the issue of *integrity*, instead relying on the semi-honest server assumption,

i.e., that the server will not deviate from the expected computation. However, many application scenarios demand more robust integrity guarantees due to the sensitivity of the data and computation involved. Beyond mere correctness concerns, the lack of integrity in FHE (which is a natural consequence of the inherent malleability of FHE ciphertexts) can also completely invalidate the confidentiality guarantees of FHE in the presence of an active adversary that might deviate from the protocol in an arbitrary and potentially malicious way [14, 17, 20, 29, 75]. For example, an active adversary can frequently recover the client's secret key and use it to decrypt all ciphertexts stored on the server, even those not used in the current computation. Since FHE, by its nature, is generally deployed in scenarios where additional data protection is critical, the potential for this protection to be suddenly negated presents a real challenge for real-world deployments. While the semi-honesty assumption can be reasonable when well supported by, e.g., legal, contractual, or reputational considerations, it clearly should not be relied upon universally. Therefore, in order to broaden the scope of applications for which FHE can be deployed, we need to consider robust FHE that is secure both against accidental corruptions and actively malicious adversaries.

Guaranteeing integrity for FHE, and specifically the correctness of the homomorphic computation, is key to achieving malicious security for FHE, as attacks on confidentiality generally require the ability to craft malicious ciphertexts. While, for some applications, this can be accomplished purely through verifying the encryption process, the vast majority of scenarios also require guarantees over the homomorphic computation itself. Note, that correct computation by itself is not technically sufficient in the malicious security setting, but, in practice, integrity mechanisms designed to achieve correctness can easily be extended to provide more general guarantees. Integrity mechanisms for homomorphic computations, however, are an inherently challenging issue, as such mechanisms (e.g., MACs, signatures, etc) generally prevent all kinds of ciphertext malleability, whereas FHE is defined by, and invariably requires, semantically meaningful ciphertext malleability. While similar issues arise in the context of maliciously-secure Multi-Party Computation (MPC), where they have been studied extensively [28], most techniques do not transfer to the FHE setting since they rely on the interactive nature of MPC (e.g., cut-and-choose protocols).

A number of works have studied the issue of integrity and/or maliciously secure FHE. For example, there has been a long series of works on IND − CCA1 secure FHE [7, 26, 27, 49, 53, 55, 68, 71], i.e., schemes that achieve indistinguishability against chosen ciphertext attacks (IND − CCA1). Unfortunately, many of these constructions assume the presence of cryptographic primitives even stronger than FHE for which no practically efficient and secure instantiations are known. A different line of research focuses purely on achieving integrity for homomorphic computations; guaranteeing a function was correctly executed on the ciphertext while preserving the confidentiality of inputs [5, 11, 12, 31, 32, 34, 36, 38]. While these are more concretely efficient than the constructions mentioned above, there is a significant gap between the assumptions made by existing work and the way state-of-the-art FHE schemes are used in practice. In particular, existing schemes can only tolerate adversaries limited to verification oracles, which are significantly weaker than the decryption oracles that frequently arise in practice. As a result, these approaches might fail to provide sufficient protection for real-world deployment scenarios.

Contributions

Currently, we lack a systematic understanding of the extent to which these approaches can be applied to common real-world deployment scenarios of FHE, how feasible it is to implement them, and the overhead that this incurs. In this paper, we provide a comprehensive analysis of existing techniques for FHE integrity. Towards this, we study and classify deployment scenarios that have been realized or are being proposed for FHE applications, and contrast them with the traditional presentation of FHE in the literature. We present a taxonomy of existing FHE integrity approaches, analyzing the assumptions and guarantees they provide in the context of complex deployment scenarios.

Based on our analysis of deployment scenarios and approaches, we conduct an experimental study of the most promising integrity techniques for FHE. In contrast to most experimental evaluations in this space, we consider modern state-of-the-art FHE schemes that offer vastly better performance but also introduce significant cryptographic and algorithmic complexity. In order to achieve this, we introduce two tools (circomlib-FHE [33] and zkOpenFHE [47]), which automate and optimize the compilation from an FHE computation to a lower-level SNARK arithmetization. We highlight the issues that arise and introduce a series of optimizations that make evaluating (e.g., SNARK-based) integrity techniques feasible in the context of these real-world schemes. We discuss the issues that arise when deploying FHE in real-world protocols, provide guidance on the state of the art of real-world FHE beyond the semi-honest setting, and identify remaining challenges and promising directions for future research towards practical robust FHE.

2 Background

We briefly introduce Fully Homomorphic Encryption and Zero Knowledge Proofs. We describe these concepts informally, highlighting properties and aspects relevant to our analysis, and refer to the extended version of this paper [48] for more formal definitions.

FHE. Fully Homomorphic Encryption (FHE) allows arbitrary computations to be performed on encrypted data. Modern FHE schemes are based on the Learning with Errors (LWE) [64] or Ring Learning with Errors (RLWE) [57] hardness assumptions. In this setting, carefully calibrated *noise* is added to the encryption. This noise grows during computation, and once it crosses a certain threshold, decryption will no longer be correct. FHE schemes address this by introducing *ciphertext maintenance* operations that do not change the encrypted data but reduce the noise (growth) in a ciphertext. FHE schemes are frequently used in *leveled* mode, where they support a parameter-dependent fixed depth of computation before the noise overflows into the message. Alternatively, *bootstrapping*, which homomorphically refreshes the ciphertext, allows for arbitrarily deep computations. However, bootstrapping is computationally expensive and, therefore, usually avoided in state-of-the-art FHE applications.

zkSNARKs. A Zero-Knowledge Sucinct Non-interactive ARgument of Knowledge (zkSNARK) is a protocol that allows a *prover* to convince a *verifier* that it knows a witness to a public instance of an NP-relation, without revealing additional information except that this instance is in the language of the relation. zkSNARKs allow the prover to generate a *proof* that the verifier can check independently. In addition, the size of this proof is sub-linear in the size of the instance and the witness. The main technical challenge in our context is *arithmetization*: translating a high-level, domain-specific property (e.g., that an encrypted logistic regression inference has been performed correctly) into a suitable instance and witness for a proof system. In particular, a widely supported and standard type of NP relation for zkSNARKs is Rank-one Constraint Systems (R1CS), where instances are triples of matrices A, B, C over a (typically cryptographically large) field, and the witness is a vector w such that $Aw \circ Bw = Cw$ (here, $\circ$ denotes the Hadamard product).

3 Why Integrity Matters

FHE applications are virtually always proposed for settings with highly sensitive data, e.g., in the medical and financial domain. While much of the focus in the literature has been on the *privacy* requirements of such applications, many works propose applications (e.g., credit rating or fraud detection) where receiving an incorrect result could be as harmful as a leak of the underlying data. However, few of these works consider the integrity of the computation, despite the fact that FHE schemes generally do not provide any integrity guarantees. In fact, FHE ciphertexts must inherently be malleable: FHE requires a server to be able to generate new ciphertexts from existing ones as part of the homomorphic computations, yet FHE schemes generally cannot guarantee that the server has indeed computed a valid ciphertext.

While detailed discussions of trust are rare in the FHE literature to-date, many papers make explicit reference to alleviating correctness issues by relying on the semi-honest server assumption (i.e., that the server does not deviate from the protocol). However, this can give the false impression that breaking the semi-honesty assumption would merely impact correctness. While correctness issues do arise, the semi-honest server assumption is in fact much more integral to FHE, as removing this assumption also impacts the *confidentiality* guarantees of FHE [14, 17, 20, 29, 75]. In the following, we first discuss how an adversary can directly exploit a lack of integrity, before discussing how this can also be used to fully undermine the confidentiality guarantees of FHE.

Correctness Issues

Fully Homomorphic Encryption inherently requires a server to be able to create new valid ciphertexts from others, i.e., FHE schemes must feature some level of *malleability*. In traditional encryption schemes, preventing malleability is generally an explicit goal, as it can introduce a variety of issues. For example, we require that tampering with a TLS channel be detectable, to prevent an attacker from changing a user's view of the internet. We can easily prevent adversaries on the network from tampering with FHE ciphertexts by communicating only via secure and authenticated channels. This does not, however, address the malleability issues that arise in the context of a server-side adversary. While relying on server-hosted applications inherently exposes one to the risk of incorrect server behavior, FHE applications usually deal with especially sensitive and important information. As a result, the potential impact of incorrect results can be significantly higher. In addition, the complexity of deploying FHE applications makes inadvertent mistakes (e.g., poor noise management leading to overflow and meaningless results) more likely and harder to distinguish from malicious behavior. Finally, there is a risk that deploying encryption might provide a false sense of security, even though harm can arise not only through leakages but also from incorrect results. For example, mispredictions on medical models might expose patients to significant health risks, and the reduced auditability of privacy-preserving systems makes such errors less likely to be identified.

A malicious server might also be incentivized to substitute a specific result that would benefit it. For example, if a client outsources the computation of a logistic regression model, the server can simply ignore the encrypted training data and instead return a model with maliciously chosen prediction behavior. Such behavior can be hard to detect, as performing the computation client-side in order to verify the result would defeat the point of outsourcing this in the first place. In addition, as FHE applications tend to be computationally expensive for the server, FHE provides an economic incentive to cheat or 'short-cut' a computation. Without additional measures in place, the server might simply choose to return a fresh (public-key) encryption of an unrelated but plausible value. Even when it is not clear what a plausible result might be, the server might choose to evaluate the function only once, and then return this result regardless of potentially differing inputs provided in future queries. Note that this behavior might also arise from implementation issues, such as an incorrect caching at any point in the software stack. While correctness issues can be problematic, they pale in comparison to the confidentiality issues that can arise from a lack of integrity.

Confidentiality Issues

Confidentiality issues in FHE can arise from the interaction between the client and the server. Specifically, the ability of an adversary to observe the behavior of the client after it decrypts the result from the server. For example, a client might query a public API with the decrypted value, essentially providing the adversary with a perfect decryption oracle. In the context of FHE, *Key-Recovery Attacks* can exploit this further to recover the secret key, therefore also compromising security for both past and future encryptions. Intuitively speaking, these attacks exploit the fact that the decryption operation combines the ciphertext and the secret key. While a decryption of a valid ciphertext will only ever output the encrypted message, a malformed ciphertext can result in (parts of) the secret key being returned instead. For example, we briefly outline a simple key recovery attack by Chenal and Tang [17] against the BV scheme [8]. In BV, decryption is defined as $(Dec)_{\mathsf{sk}}(\mathbf{ct}) = [\mathbf{ct}_0 + \mathbf{ct}_1 \cdot \mathsf{sk}]_t$, where the inner operations are performed over R_q (see [8] for details). When decrypting the special ciphertext $\mathbf{ct} = (0, 1)$, one trivially recovers the secret key $(Dec)_{\mathsf{sk}}(\mathbf{ct}) = [0 + 1 \cdot \mathsf{sk}]_t = [\mathsf{sk}]_t = \mathsf{sk}$ (under some assumptions on the parameters; we refer to [17] for the details). In this simple attack, the client can easily detect that this ciphertext has been maliciously crafted.

However, FHE also features encryptions of the secret key that are indistinguishable from standard ciphertexts. For example, key-recovery attacks can take advantage of the *evaluation keys* provided to the server in most schemes. These are encryptions of (functions of) keys, which allow the server to perform crucial ciphertext maintenance operations (e.g., relinearization, key-switching, bootstrapping). If given access to a decryption oracle, an adversary could decrypt these evaluation keys and recover the secret key. In the (approximate) CKKS [19] scheme, which inherently intermingles (key-dependent) noise with the message, such one-shot key-recovery attacks are even possible given the decryption of *any* ciphertext [51]. In a similar setting, key-recovery attacks are also possible in practice for "exact" FHE schemes, due to their noticeable probabilities of decryption error [18].

Full decryption oracles that would allow an adversary to exploit the straightforward attacks we discussed above are thankfully not commonplace in practice. However, adversaries can also exploit more subtle *reaction* oracles, which arise much more naturally, even in well-designed applications. For example, contact tracing applications necessarily result in different behavior from the client depending on whether a contact was detected. Sophisticated attacks that exploit these reaction oracles have been shown to be very practical for all modern FHE schemes [20]. Essentially, these attacks rely on the ability to create a ciphertext with large, but carefully chosen, amounts of noise, so that during decryption, the result will either overflow the plaintext modulus or not, depending on the concrete value of the secret key. Through a series of such oracles, the adversary can iteratively recover bits of the secret key, eventually recovering the full key. Because of the ability to batch many different plaintexts into a single ciphertext in most modern FHE schemes, the actual number of queries required to fully recover a key this way can be extremely small and is primarily limited by the quality of the reaction oracle. In addition to manipulations from a malicious server, reaction oracles might also arise when a client inadvertently chooses a circuit that causes noise overflow and therefore has key-dependent behavior, which gives rise to the same oracles even in a semi-honest setting. Most literature on FHE implicitly assumes that circuits correctly realize the desired plaintext functionality, yet in practice, ensuring that this hidden assumption is always fulfilled is non-trivial [22, 23].

While it is possible to reduce the attack surface for reaction oracles through careful protocol design, observable reactions cannot generally be fully avoided, as it would prevent the client from acting upon the results of the computation in any way. As a result, violations of the semi-honest server assumption are not only a correctness issue, but can virtually always fundamentally undermine the confidentiality of sensitive data in practice. Therefore, such settings require FHE to be augmented with techniques that provide integrity and malicious security, preventing an actively malicious server from crafting the malformed ciphertexts that enable them to exploit client reactions. However, addressing FHE integrity in an actively malicious setting is both inherently challenging and depends significantly on the topologies of trust in a given deployment.

4 Taxonomy of Integrity Approaches

A variety of approaches towards FHE Integrity has been proposed, but we currently lack a systematic understanding of how expressive they are, the assumptions they make and the guarantees they provide, and, as a result, their applicability to the different real-world deployment topologies we identified. In this section, we aim to provide a clearer understanding of existing approaches and their capabilities. Towards this, we provide a taxonomy of existing approaches, categorizing them by the underlying integrity primitives they employ. In Section 5, we introduce tools that implement and automate the most promising of these approaches, along with efficiency and useability optimizations. We then follow this up with an experimental evaluation of the most promising solutions in Section 6 to assess practical considerations.

Scope

Ciphertext malleability is a defining aspect of FHE, as it is necessary to allow homomorphic computations to be executed over encrypted data. However, as we discussed in Section 3, this malleability is also the root cause of a series of integrity and security issues. Traditional approaches to integrity aim to completely eliminate malleability (e.g., authenticated encryption with associated data, message authentication codes, or signatures) and are therefore not suitable to secure computation. This issue of selective malleability also arises in the context of secure Multi-Party Computation (MPC). However, most techniques do not transfer to the FHE setting since they rely on the interactive nature of MPC (e.g., cut-and-choose protocols). Cryptographically guaranteeing the correct execution of computations is never trivial, but modern Zero Knowledge Proofs are able to prove the correctness of complex functionalities efficiently and succinctly. However, the encrypted nature of FHE requires additional considerations, and we therefore only consider FHE-specific techniques.

There have also been several proposals for application-specific approaches to achieving stronger integrity guarantees and/or malicious security in the context of FHE. For example, tailored proof systems have been proposed for the use of (F)HE when used as specialized building blocks in larger protocols (e.g., to generate multiplication triplets for secret-sharing-based secure multiparty computation [44]). Alternatively, some applications use additional 'blinding' on the messages they encrypt so that even a full compromise of the FHE system does not give immediate access to the underlying values [16]. Related approaches focus on proofs of correct encryption [13, 25, 54] or decryption [10, 56] for FHE ciphertexts. These approaches are frequently highly efficient, as they can be tailored specifically to the required setting, both in terms of required guarantees and in terms of implementation. However, as these techniques generally do not transfer to more general settings, we consider them out of scope for this work. In a similar vein, we exclude approaches that rely on primitives or constructions for which no efficient and secure instantiations are known (e.g., relying on indistinguishability obfuscation [72]).

We categorize the different approaches into three groups based on the underlying integrity primitives they build upon: Message Authentication Codes (MAC), Zero-Knowledge Sucinct Non-interactive

ARgument of Knowledges (zkSNARKs), or Trusted Execution Environments (TEE). In the following, we discuss the approaches in each category in more detail, highlighting advantages and shortcomings.

Message Authentication Codes

While Message Authentication Codes (MACs) have long been used to prevent any malleability in ciphertexts in the context of traditional symmetric-key encryption, *homomorphic* MACs are fundamentally different constructions and share little beyond their symmetric nature with their namesakes (i.e., a secret MAC-specific key is required both to generate and verify them). Homomorphic MACs are essentially tags that are associated with either the plaintext or ciphertext (depending on the specific approach) and are processed alongside them by the server. The client then recomputes the (plaintext) function the server (allegedly) evaluated, but only over these tags, which - if done correctly - can significantly lower the computational effort required.

As a consequence of their symmetric nature, MAC-based approaches are fundamentally incompatible with scenarios that require (private) inputs from more than one party, i.e., they only support the outsourcing setting. In this setting, they provide strong guarantees for both correctness and confidentiality. While they are usually proposed against a slightly weaker adversary model, the restricted setting they operate in means that they also achieve security against stronger IND − CCA1 adversaries trivially. Similar to traditional MACs, homomorphic MACs can be designed to work in three different modes, which impact practical aspects such as their expressivity.

Encrypt-then-MAC. The Encrypt-then-MAC (EtM) approach firsts encrypts the plaintext using FHE and then applies the MAC to the resulting ciphertext. Therefore, the MAC does not need to offer confidentiality guarantees. However, this requires the MAC to be homomorphic with respect to operations on *ciphertexts*, which notably seems even more complex than the homomorphism over plaintexts offered by FHE. For example, Fiore et al. make use of this paradigm in [31], instantiating their MAC using pairings. In order to enable some form of homomorphism, they introduce a homomorphic hash function to bridge the gap between FHE ciphertexts and the MACs, i.e., polynomial rings and the pairing groups. However, the combination of primitives they rely on limits the expressiveness of the resulting construction. Specifically, it only supports computations with at most one multiplication gate. Currently, it is unclear whether it is possible to create MACs that support both arbitrarily deep circuits and complex operations on the ciphertexts.

Encrypt-and-MAC. In the Encrypt-and-MAC (EaM) paradigm, the initial MAC and FHE ciphertext are computed from the same plaintext and are then processed in parallel (but independently) to produce the output MAC and ciphertext pair. Since the MAC in EaM is not encrypted under FHE, it must itself provide strong security, i.e., be semantically secure. In addition, the MAC must offer the same homomorphic operations as the underlying FHE scheme. Li et al. construct such an EaM scheme using multilinear maps [52]. While it describes how to support addition and multiplication operations, it is not clear how this scheme can be extended to handle the complex ciphertext maintenance operations (e.g, relinearization) that are necessary for modern FHE schemes to achieve state-of-the-art efficiency. In general, it is unclear how to expand the expressiveness of homomorphic MACs while maintaining the strong security guarantees required for the EaM approach.

MAC-then-Encrypt. The MAC-then-Encrypt (MtE) paradigm first computes a MAC over the plaintext and then encrypts the MAC-augmented plaintext under FHE. Gennaro and Wichs [38] provided one of the first FHE integrity construction based on this paradigm. However, the construction is not efficiently verifiable, i.e., verifying the MAC requires recomputation that is as expensive as computing the original result. The work proposed potential ways to solve this issue, but did not instantiate a solution. Catalano and Fiore [11] addressed the verification efficiency, but in turn their construction is limited to arithmetic circuits of a bounded depth. More recently, Chatel et al. [12] have generalized these two approaches to achieve efficient verification for arbitrary circuits, providing the first practical FHE integrity scheme that can efficiently support arbitrary circuits and modern state-of-the-art schemes, if only in the outsourced setting.

zkSNARKs

zkSNARKs allow a prover to convince a verifier of the truth of a statement with a non-interactive and succinct proof, and without revealing additional information. In the context of FHE, the server would compute the FHE circuit as normal, storing any (encrypted) intermediate results, and then compute a Succinct Non-interactive ARgument of Knowledge (SNARK) that asserts that the server knows an assignment of intermediate values to the circuit so that, for the given input, the circuit results in the output ciphertext. While existing literature frequently suggests combining FHE with proof systems in order to strengthen its circuit guarantees, a straightforward combination does not, in fact, achieve security against an active adversary. Instead, we need to augment this approach with techniques such as the Naor-Yung Construction [59] that achieve not only correctness but also IND − CCA1 security.

Using *zero-knowledge* proofs (zkSNARK), the server can provide the required correctness guarantees while maintaining the privacy of its own inputs. This allows this approach to handle the two- and multi-party computation settings. However, the latter might require either the use of publicly verifiable proofs in order to allow all clients to verify the proof, or the creation of separate proofs for each of the clients. When extended appropriately, ZKP-based techniques can achieve strong security guarantees against active adversaries in all settings. However, different proposed solutions differ dramatically in their expressiveness with regard to input checks and the underlying operations of modern state-of-the-art FHE schemes.

FHE-specific zkSNARKs. Recent work has focused on developing proof systems tailored to FHE. One line of work uses (homomorphic) hashing to bring the size of FHE ciphertexts down into a range that can be handled more efficiently with ZKP techniques. This includes the first SNARK for FHE presented by Fiore et al. [32] and follow-up work by Bois et al. [5]. However, the homomorphic hashing requirement limits this approach to simple schemes such as the BV scheme [8] which does not feature the complex

ciphertext maintenance operations that are necessary to achieve the practical efficiency enjoyed by state-of-the-art FHE schemes. In addition, there has been work on more limited proof systems designed specifically to prove the correct encryption [13, 25, 54] or decryption [10, 56] for FHE ciphertexts. These support modern FHE schemes, but the techniques used generally do not transfer to the significantly more complex challenge of proving FHE computation correctness. Nevertheless, they can be an integral part of overall solutions, e.g., in the multi-party setting, where proofs of encryption are required for all inputs.

Ring-SNARKs. Rather than focusing on SNARKs specifically for FHE schemes, Ganesh et al. [34] propose an alternative approach that focuses on proof systems for the Ring algebra used by many FHE schemes. While the initial work relied on an inefficient encoding to guarantee security, we show in Section 5 how to optimize these to achieve a significantly more practical system. The ring-based SNARK drastically improves the efficiency of proving the operations that make up basic homomorphic operations such as addition and multiplication. However, it cannot express ciphertext-maintenance operations which frequently require either switching between different rings or non-ring operations such as rounding. Without support for these, proofs are limited to small (specifically, shallow) circuits. In addition, their ring-based nature does not offer a way to efficiently express input constraints (which, again, are frequently about properties beyond the ring algebra). As a result, Ring-SNARKs (in their current) form are limited to (simple) outsourcing settings.

Generic zkSNARKs. A variety of works simply suggests combining FHE with generic ZKP techniques in order to address FHE integrity. However, actually realizing such solutions in practice faces significant challenges. While solutions such as zero-knowledge Virtual Machines (zkVMs), proof systems that offer the ability to run and prove nearly unmodified software, could provide an easy path to implementation, the performance is prohibitive for even the most simple applications [6]. Using more efficiency-focused traditional proof systems, however, requires arithmetizing complex homomorphic operations into the native algebra over which these proof systems express their constraints, which is non-trivial.

Trusted Execution Environment Approaches

Trusted Execution Environments (TEEs) are hardware components capable of isolating code running on them from the rest of the machine, even the operating system or hypervisor. TEEs are commercially available in commodity hardware provided by all major hardware vendors, especially when considering server platforms. While TEEs can be used to provide confidentiality, a series of attacks [30, 62] has put their suitability for this task in question. However, their integrity protections, i.e., their ability to *attest* to the program running in the enclave, have so far mostly resisted practical attacks [58].

TEEs can support a wide variety of deployment settings, assuming the necessary hardware is available. Specifically, they can easily express required input checks in the two- and multi-party settings through simple code; attestation could even be used by the input parties to provide guarantees of correct encryption in the MPC setting. While it might be difficult to compare the guarantees derived from trusted hardware with traditional cryptography approaches, TEEs can be used, with trivial extensions to the work described in the literature, to achieve protections against a fully malicious server, assuming the server cannot compromise (the attestation component of) the secure enclave.

FHE-in-TEE. TEE-based solutions are conceptually much simpler, and essentially revolve around the challenge of realizing the complex FHE implementations inside the frequently restrictive enclave environment. Natarajan et al. [60] present an FHE-in-TEE design that ports the Microsoft SEAL library [21] to Intel SGX. As a result, it fully supports a variety of state-of-the-art FHE schemes. Existing work uses a traditional SDK-based porting strategy [63], but more recent approaches to realizing software in SGX can support virtually unmodified binaries of existing code [15]. This not only improves usability, but is also expected to provide improved performance. However, this approach is less portable between different vendors.

In this work, we focus on cryptographic integrity approaches, as they are the most widely applicable and provide the strongest guarantees. Out of these, we specifically focus on general-purpose SNARKs to benefit from their strong guarantees, expressivity (both for supporting different FHE schemes, as well as additional input checks), and flexibility in the underlying proof system (and various efficiency tradeoffs). In the next section, we introduce two software frameworks that implement and automate the arithmetization of FHE operations, along with efficiency and usability optimizations.

5 From Theory to Practice

Generic proof systems are the most promising and flexible tool for adding integrity to FHE pipelines. However, this theoretical promise has not yet been realized in practice, and is hampered by three main challenges: (i) it is technically non-trivial to arithmetize FHE schemes for proof systems, (ii) a significant amount of optimization taking advantage of the specific properties of FHE schemes is required to improve efficiency, and (iii) the resulting software artefact needs to be accessible and easily useable by practitioners.

In the remainder of this paper, we address these three challenges in turn, presenting a comprehensive implementation, several optimizations, and a framework to augment existing FHE applications with integrity guarantees and to foster further research in this area.

zkOpenFHE

In order to provide a solid foundation for FHE integrity research, we introduce zkOpenFHE, a framework that augments the OpenFHE library with ZKP-based integrity guarantees. We chose OpenFHE as the basis for our work because it is a comprehensive and modular library that supports a wide range of FHE schemes (B/FV, BGV, CKKS, FHEW, TFHE). zkOpenFHE's application programming interface (API) is a superset of OpenFHE's API, allowing users to easily add integrity guarantees to their existing FHE applications. Starting with a program that uses OpenFHE to perform computations on encrypted data, the user must additionally specify which are the inputs and outputs of the computation. zkOpenFHE provides three modes of operation:

EVALUATION: In this mode, zkOpenFHE simply leverages OpenFHE to run the encrypted computation with no integrity guarantees and no additional slowdown. This mode is intended primarily for prototyping, debugging, and testing.
CONSTRAINT_GENERATION: zkOpenFHE only considers the circuit of the encrypted computation, and automatically arithmetizes it (while applying the local and global optimizations described in the next sections) to generate the corresponding constraints. This mode is intended as a one-time compilation step (both for the prover and verifier). Along with the constraint, zkOpenFHE also generates a mapping from the FHE circuit to individual variables for the constraint system. For proof systems with a circuit-specific setup phase, the output of this mode can be used as input to the setup phase in order to generate the common reference string.
WITNESS_GENERATION: In this mode, zkOpenFHE uses OpenFHE to run the encrypted computation, and uses the constraints and circuit-to-variable mapping generated in `CONSTRAINT_GENERATION` mode to automatically generate a satisfying witness for the constraint system. zkOpenFHE outputs the result ciphertext of the FHE computation, along with the witness. This witness can then be used with zkOpenFHE's built-in Groth16 [41] prover to generate a proof, or it can be exported to be used with an external proof system.

Rinocchio

Among the FHE-friendly proof systems, only Rinocchio [34] by Ganesh et al., is capable of expressing (parts of) modern FHE schemes. However, its expressiveness remains limited, as Rinocchio only supports arithmetic ring operations, whereas some FHE operations (e.g., relinearization) use component-wise rounding operations internally. As a result, we are severely limited in the complexity of the circuit we can use with Rinocchio. As presented, Rinocchio uses a highly inefficient encoding system, and we extend it with a more optimized encoding scheme, which we discuss below. Additionally, Rinocchio only provides around 60 bits of (computational) soundness for the rings used in FHE. We use a simple soundness amplification strategy, running three separate instances of the protocol to achieve stronger soundness guarantees.

Optimizing Rinocchio. In the following, we describe our optimizations for the Rinocchio protocol by Ganesh et al. [34]. The original paper introduces 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.

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 $R_{q_1}, \ldots, R_{q_l}$ (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 $l \cdot \log_q(Q)$, improving on the Regev encoding by a factor of n. Using this batching technique imposes the requirement $q_i = 1 \mod 2N$ on the ciphertext moduli of the FHE scheme; however, this condition is already necessary for some schemes (e.g., RNS-optimized BGV [45]), and can be easily satisfied for all other schemes. We provide an open-source implementation [46] of this optimization of the Rinocchio SNARK.

General-Purpose SNARKs

State-of-the-art proof systems are highly efficient, but have primarily been tailored to applications that share few characteristics with FHE. Expressing modern state-of-the-art FHE schemes introduces a series of challenges not just because homomorphic operations, especially ciphertext maintenance operations are computationally complex, but also because FHE and proof systems operate on fundamentally different types of algebras. The Rinocchio scheme [34] is based on Ring-Learning with Errors (RLWE) and uses rings of the form $R_q := \mathbb{Z}_q[X]/\langle X^N + 1\rangle$, i.e., polynomials with degree up to N (usually $N > 2^{13}$) and coefficients in $\mathbb{Z}_q$. Most proof systems, on the other hand, mostly use large prime fields (i.e., $\mathbb{Z}_p$ where p is usually a 254-bit prime). When using modern NTT-based implementations of FHE, we do not need to consider the polynomial reduction modulo $X^N + 1$, however, we must still address the mismatch between the coefficient modulus q and the field modulus p. Existing works frequently propose instantiating the FHE and proof systems so that $q = p$, however, this is only possible for toy FHE schemes as both efficient FHE and proof systems impose significant restrictions on the moduli, frequently leaving no overlap. While FHE security benefits from limiting the size of q, the security of the proof system often requires sufficiently large p, introducing an inherent conflict. Additionally, modern FHE schemes generally use the Chinese Remainder Theorem (CRT) to decompose the coefficient modulus q into L even smaller moduli $q_1, \ldots, q_L$, working on each *Residue Number System* (RNS) limb independently to improve performance. As a result, matching the proof system and FHE moduli is almost always infeasible for practical applications.

Given a computation defined as a circuit, different proof systems follow different approaches in translating the correctness of computation into an arithmetic proof system. We use R1CS, one of the most widespread arithmetization approaches, which converts circuits into a system of Rank-1 constraints. Basic arithmetic operations such as additions and multiplications can be realized directly using a single constraint. However, more complex operations (e.g., rounding) require a larger number of constraints. Since the majority of operations in (NTT-based) FHE are simple arithmetic operations, they can be efficiently translated to R1CS. Beyond R1CS, STARKs [4] present an alternative way to express computations not as circuits

but as a series of data manipulations, in the so-called Algebraic Intermediate representation (AIR). The efficiency of STARKs is directly related to the size of the state space and the complexity of the transition function describing each step. Since FHE features large ciphertext expansion and a high-degree modulo function, this makes it a poor candidate for realization using STARKs. One can also forgo explicit arithmetization and instead directly express arithmetic circuits. However, this approach scales poorly with the number of inputs, which is high for FHE circuits due to the expansion from individual ciphertexts to $2N$ field elements (where $N \geq 2^{13}$). Finally, generalizations of R1CS have also been proposed (such as Plonk-ish and CCS relations); however, these do not immediately seem to offer a tangible benefit for FHE arithmetization (in particular, arithmetizing modular reductions leads to custom gates with very high degree, which are inefficient to prove). We focus on R1CS for this first version of zkOpenFHE, but our modular framework allows for easy extension to other constraint systems as they become more generally available.

Emulating modular arithmetic. We use a *modulo emulation* method that bridges the modulus gap. This allows us to choose $q < p$, which complements each system's security requirements while requiring them to match only the constraints of their own system. We can then *emulate* operations modulo q_i in the proof system's (mod p) field, by explicitly computing the modular reduction mod q_i after each arithmetic operation. However, proving the correctness of this modular reduction (which is necessary for soundness to be meaningful) requires two expensive range proofs. As, in practice, $p \gg 2q$, we propose an optimization based on *lazy* modulo emulation. Because modular reduction produces the same result whether it is applied to the inputs or the outputs of an operation, we can wait until just before the results could overflow and only compute and prove the modulo reduction then. Specifically, we can perform $\left\lceil \frac{\log_2 p}{\log_2 q} \right\rceil$ multiplication operations in sequence before needing to reduce. For small circuits with $\log_2 q \approx 60$ bits and a standard field-based proof system with $\log_2 p \approx 254$ bits, this enables a 4× reduction in the number of modulo operations.

Without this optimization, there is little difference between the RNS and non-RNS approaches with respect to the effort required to prove modulus gates. This is because the cost of proving a modular reduction is roughly linear in the bit-width of the modulus. However, with this optimization, using the RNS approach allows us to reduce the number of modular reductions even further. Considering an FHE circuit with k arithmetic operations, a non-RNS, non-optimized implementation requires k modular reductions of size $\log_2 q$. This optimization reduces this to roughly $k\frac{\log_2 q}{\log_2 p}$ modular reductions of size $\log_2 q$. By RNS-splitting each element into L limbs, the size of each gate is reduced to $\log_2 q_i \approx \frac{\log_2 q}{L}$ but, without any optimizations, the number of modular reductions increases to kL, negating the benefits. However, with this optimization, we require only $kL\frac{\log_2 q_i}{\log_2 p} \approx k\frac{\log_2 q}{\log_2 p}$ modular reductions of reduced size $\frac{\log_2 q}{L}$. Therefore, with this optimization, RNS allows us to reduce the cost per modular reduction even while already reducing their number. For example, with $\log_2 q \approx 60$ bits and $\log_2 p \approx 254$ bits as above, an RNS approach splitting q into two 30-bit moduli would halve the cost again, giving us a total 8× decrease in modular reduction overhead.

circomlib-fhe

We have implemented a comprehensive optimization of RLWE and LWE FHE schemes in the high-level domain-specific circom language. This library, circomlib-FHE, is designed to be used as a prototyping library for the arithmetization of FHE schemes. While circom does have support for various proving backends, circomlib-FHE is not designed to be directly applied to prove and verify an encrypted computation. Rather, it is designed to be used as a prototyping tool to quickly and efficiently test and optimize arithmetizations for FHE schemes in the context of proof systems. We make circomlib-FHE available as an open-source library [33] to foster further research in this area.

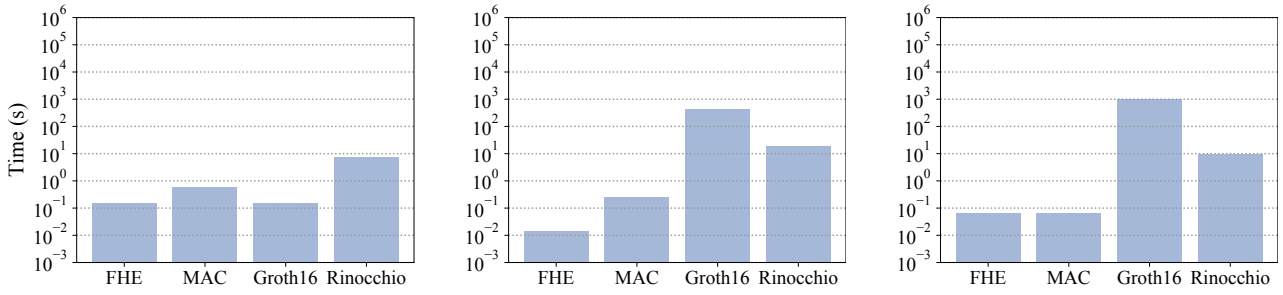
6 How efficient is FHE with Integrity?

Leveraging circomlib-FHE and zkOpenFHE, in this section we aim to give of sense of the slowdown introduced by adding integrity to existing FHE pipelines. First, we compare the runtime of different approaches to FHE integrity (see Section 4). In Section 6.2, we then focus on the promising approach of general-purpose proof systems, and examine which building blocks of FHE schemes are most expensive to arithmetize, and how different FHE schemes compare in terms of constraints.

6.1 Comparing primitives

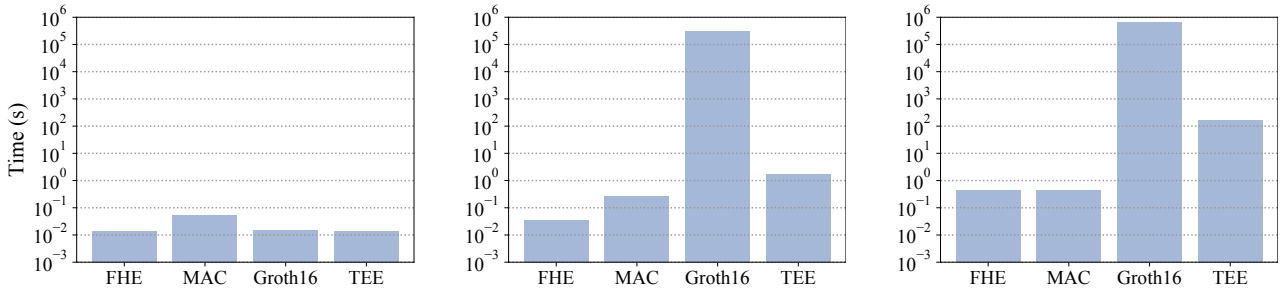
In order to provide an understanding of the concrete costs of different approaches to FHE integrity, we implement and evaluate a variety of different approaches. We consider homomorphic MACs [12], FHE-in-TEE [60], Ring-SNARKs [34] and an instantiation of a generic proof system [37] with Groth16 [41]. While real-world FHE applications frequently fall into the two-party or multi-party setting, we restrict our evaluation to the outsourcing setting, as several techniques do not support further settings [12, 34]. Considering the outsourcing setting allows us to have a more direct comparison, and, as we will show, already presents a significant challenge for current solutions. Because Ring-SNARKs [34] are not only limited to the outsourced settings, but also do not support ciphertext maintenance operations, we consider two different workloads, both aimed at logistic regression inference. In the standard workload, we consider a setting with polynomial degree 8192, 32 bit plaintext modulus and a more complex circuit for 512 features using a degree eight polynomial approximation for the sigmoid operation. In the simplified workload, we consider a setting with polynomial degree 2048, 16 bit plaintext modulus and a simplified circuit for 64 features using a degree two polynomial approximation for the sigmoid operation.

Implementation & Setup. For the homomorphic MACs, we re-use the original implementation by Chatel et al. [12]. For the TEE-based approach, we use Gramine [15] to encapsulate and attest an otherwise unmodified FHE implementation (specifically, we use the Microsoft SEAL [21] library) running in a Docker container. For the ring-SNARK and the generic SNARK approach, we use our optimized implementation of Rinocchio and our zkOpenFHE library, respectively. We evaluate our implementations on an AWS



(a) Client **(b) Server** **(c) Setup**

Figure 1: Runtimes for simplified logistic regression inference, comparing a pure FHE baseline with integrity approaches.



(a) Client **(b) Server** **(c) Setup**

Figure 2: Runtimes for standard logistic regression inference, comparing a pure FHE baseline with integrity approaches.

`c5.9xlarge` instance with 32 vCPUs and 72 GB of RAM, with the exception of the TEE-based implementations, where we use an Azure `DC16s v3` instance with 16 vCPUs and 32 GB of RAM, as this provides the necessary Intel SGX enclave.

In Figure 2, we present the client- and server-runtimes for the standard logistic regression inference workload. We contrast MAC, SNARK, and TEE-based integrity approaches with a baseline of "plain" FHE without integrity guarantees. Note, that this is an artificial workload, i.e., the encryption of the inputs already requires significantly more client-side computation than evaluating the underlying function directly on the client would require. Nevertheless, it allows us to investigate the overhead of different FHE integrity approaches. While all techniques introduce significant overhead for the server, there is a vast gap between the overhead introduced by MACs and TEEs and the overhead introduced by a solution using generic proof systems - even after our significant optimizations to reduce the number of constraints required. In the TEE setting, the server is realized via Docker containers, introducing a base overhead that is not present in the other (non-containerized) approaches. For more real-world sized workloads, however, this overhead should amortize to no more than 1 order of magnitude over the FHE baseline, and in practice, frequently much lower overheads (e.g., 2-4x [15, 60]).

The standard version of our workload fully utilizes the SIMD (Single-Instruction Multiple Data) parallelism present in RLWE-based FHE schemes by encrypting data corresponding to different features into the different slots of a single ciphertext. This (latency-)optimized approach is common in virtually all state-of-the-art FHE workloads using RLWE-based FHE schemes, but requires homomorphic rotations, which cannot be expressed in Ring-SNARKs. In Figure 1, we therefore consider our simplified workload which can be expressed by Ring-SNARKs, contrasting it against generic proof systems, MACs and an unprotected FHE baseline. The simplified version considers not only a significantly smaller problem size, but also switches from latency-optimized batching to a simplified encoding, where each feature is encrypted into its own ciphertext. As we can see, Ring-SNARKs offer a noticeable reduction in server-side overhead compared to generic proof systems, but still introduce significant overheads. In addition, they also increase client-side verification effort compared to the extremely efficient Groth16 SNARK.

Overall, while we demonstrate that it is possible to run non-trivial FHE circuits while providing strong integrity guarantees, it is clear that SNARK-based solutions are, as of now, far from being practical for complex workloads.

6.2 R1CS Constraints

While the results of the previous section give a good sense of the overhead introduced by different integrity primitives in the restricted outsourcing setting, SNARK-based approaches are the only cryptographic approach that provide meaningful integrity guarantees in more complex deployment settings. In addition, the concrete runtime of a SNARK-based approach varies with the proof system used (e.g., Groth16 provides constant and concretely small proof

Scheme	ModSw.	KeySw.	NTT	SignDec.	Ext. Prod.
	103 525	16 297 984	179 728	591 872	133 120
FHEW	0.04%	0.65%	73.01%	21.63%	4.66%
TFHE	0.07%	1.08%	72.94%	18.01%	7.90%

Table 1: R1CS constraints for common building blocks FHE schemes (modulus-switching, key-switching, NTT, signed digit decomposition, RLWE-RGSW external product).

Scheme	Accumulator			Bootstrapping
	Init	Update	Extract	
FHEW	1 233 936	2 566 200 000	359 456	2 584 298 426
TFHE	1 233 936	1 500 651 008	359 456	1 518 749 434

Table 2: R1CS constraints for accumulator phases and a single programmable bootstrapping operation.

size and verification runtime, at the expense of a trusted setup and log-linear prover runtime). In this section, we therefore focus on the number of constraints required to arithmetize different building blocks of FHE schemes, as this directly influences the runtime of a SNARK-based approach. We focus on the FHEW and TFHE schemes, as they allow us to study the behaviour of more complex building blocks (in particular, a full bootstrapping operation) with lower complexity than other schemes (e.g., B/FV, BGV or CKKS).

Table 1 shows the number of constraints for common FHE sub-operations, as well as their relative contribution to the total number of constraints required for a bootstrapping operation (using both the FHEW and TFHE scheme). These numbers were obtained by automatically arithmetizing the respective operations using the circomlib-FHE library. Key-switching proves to be the most expensive operation, followed by signed digit decomposition and NTT. However, since NTTs are used much more frequently, they total $> 70\%$ of the constraints required for bootstrapping, in both FHEW and TFHE.

Table 2 shows the contributions of the different phases of an accumulator (initialization, update, and extraction) towards a single bootstrapping operation. Both FHEW and TFHE require more than a billion constraints for a single bootstrapping operation, with the vast majority of constraints being required for the update phase. This number of constraints is larger than what typical proof systems can handle (e.g., without running out of memory), and actually proving this relation would require a more sophisticated proof system that can handle such large constraint sets. Nevertheless, most of these constraints stem from NTTs (interestingly, FHE runtimes are also dominated by NTTs), and progress towards a better NTT arithmetization would immediately yield significant improvements. RLWE-based schemes such as B/FV or BGV require fewer constraints for low-depth circuits (since they do not require bootstrapping). Nevertheless, NTTs also contribute the most towards the total number of constraints.

7 Discussion

Fully Homomorphic Encryption is emerging as a promising solution for real-world privacy-preserving systems. However, as we transition from proofs of concept to actual deployment, we must consider the issue of integrity more carefully. This is especially true when considering adversarial settings beyond the semi-honesty assumption that underlies the vast majority of current work on FHE. While the implications of this assumption are well understood in the cryptographic community, there seems to be a lack of consideration for the implications that violating this assumption has on practical deployments. We hope that this paper serves to bring this issue to wider attention in the community and encourage more careful consideration of integrity in FHE systems.

FHE integrity has, so far, virtually always been "addressed" via trust assumptions, be they direct (semi-honest server assumption) or indirect (hardware trust assumption for TEEs). While a variety of solutions based on cryptographic hardness assumptions have been proposed, these struggle to support non-trivial applications effectively. We have seen a variety of works that propose solutions that fall short of achieving malicious security or only consider impractically restrictive adversarial models. In addition, of the existing solutions, a significant number focuses primarily on the outsourcing setting, which is not sufficient to enable the vast majority of actual and proposed FHE-based privacy-preserving applications. In general, our understanding of malicious security in the context of FHE lags significantly behind similar considerations in the fields of secure multi-party computation. This is likely not least due to the significant additional challenge for integrity solutions that FHE presents due to the noisy nature of FHE encryption and computation. Clearly, there is a significant need for further exploration, but also increased education and communication both to guide future research towards relevant settings and to make practitioners aware of these considerations.

In this paper, we shed light on the inherent complexity of cryptographically ensuring the integrity of FHE computations, specifically in the context of more complex topologies of trust which appear in many real-world deployment scenarios. We provide a systematic analysis of existing work considering the assumptions they make and guarantees they provide in the deployment settings we identify. We augment our analysis with an experimental evaluation of the most promising approaches to assess the current practicality of providing integrity protections for FHE. Even with our (sometimes significant) optimizations, we show that most approaches fall considerably short of being widely applicable in practice due to the severe performance overhead of generic techniques. Unfortunately, solutions targeted specifically at FHE that promise to alleviate these performance issues fall short in their expressiveness and/or applicability to real-world deployment scenarios and therefore do not currently present a practical alternative.

Given the current state-of-the-art, work on deploying FHE should currently proceed cautiously and ensure that it can either fully justify a semi-honesty assumption or rely on hardware trust in order to ensure integrity. While there remain more than enough application scenarios for which these are workable solutions and which are yet to be explored with FHE, this clearly poses a noticeable restriction on the possible applications of FHE. As a result, we believe the community needs to increase its efforts into achieving practical integrity guarantees for FHE. Most likely this will need to follow

the FHE+zkSNARK approach, however, there are multiple paths forward in this area. Results might be achieved both by improving the efficiency of emulating FHE in existing proof systems, or by unlocking the potential of custom FHE-friendly proof systems by co-designing them with modern state-of-the-art schemes in mind. With zkOpenFHE, we propose a framework that we hope can act as a solid foundation for such future research into verifiable FHE.

Acknowledgments

We would like to thank Chaya Ganesh, Anca Nitulescu, Kenny Paterson, Eduardo Soria-Vazquez, Michael Steiner, Nojan Sheybani, Erin Hales, Lea Nürnberger, Martha Norberg Hovd, and the Privacy-Preserving Systems Lab team at ETH Zurich for their insightful input and feedback. We would also like to acknowledge our sponsors for their generous support, including Meta, Google, SNSF through an Ambizione Grant No. 186050, and the Semiconductor Research Corporation.

References

[1] Yongdae An, Seungmyung Lee, Seungwoo Jung, Howard Park, Yongsoo Song, and Taehoon Ko. 2021. Privacy-Oriented Technique for COVID-19 Contact Tracing (PROTECT) Using Homomorphic Encryption: Design and Development Study. *J. Med. Internet Res.* 23, 7 (July 2021), e26371. http://dx.doi.org/10.2196/26371

[2] Apple Inc. 2024. Announcing Swift Homomorphic Encryption. https://swift.org/blog/announcing-swift-homomorphic-encryption/.

[3] Apple Inc. 2024. Getting up-to-date calling and blocking information for your app. https://developer.apple.com/documentation/sms_and_call_reporting/getting_up-to-date_calling_and_blocking_information_for_your_app.

[4] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2018. Scalable, transparent, and post-quantum secure computational integrity. *Cryptology ePrint Archive* (2018). https://eprint.iacr.org/2018/046.pdf

[5] Alexandre Bois, Ignacio Cascudo, Dario Fiore, and Dongwoo Kim. 2021. Flexible and Efficient Verifiable Computation on Encrypted Data. In *Public-Key Cryptography – PKC 2021*. Springer International Publishing, 528–558. http://dx.doi.org/10.1007/978-3-030-75248-4_19

[6] Emiliano Bonassi. 2024. GitHub - emilianobonassi/zkFHE: Verifiable and confidential computation based on ZKP and FHE, powered by risc0 zkVM. — github.com. https://github.com/emilianobonassi/zkFHE [Accessed 2024-07-26].

[7] Dan Boneh, Ran Canetti, Shai Halevi, and Jonathan Katz. 2007. Chosen-Ciphertext Security from Identity-Based Encryption. *SIAM J. Comput.* 36, 5 (Jan. 2007), 1301–1328. https://doi.org/10.1137/S009753970544713X

[8] Zvika Brakerski and Vinod Vaikuntanathan. 2011. Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In *Advances in Cryptology – CRYPTO 2011*. Springer Berlin Heidelberg, 505–524. http://dx.doi.org/10.1007/978-3-642-22792-9_29

[9] Rosario Cammarota. 2022. Intel HERACLES: Homomorphic Encryption Revolutionary Accelerator with Correctness for Learning-oriented End-to-End Solutions. In *Proceedings of the 2022 on Cloud Computing Security Workshop* (Los Angeles, CA, USA) *(CCSW'22)*. Association for Computing Machinery, New York, NY, USA, 3. https://doi.org/10.1145/3560810.3565290

[10] Christopher Carr, Anamaria Costache, Gareth T Davies, Kristian Gjøsteen, and Martin Strand. 2018. Zero-knowledge proof of decryption for FHE ciphertexts. *IACR Cryptol eprint Arch* 2018 (2018), 26. https://eprint.iacr.org/2018/026

[11] Dario Catalano and Dario Fiore. 2013. Practical Homomorphic MACs for Arithmetic Circuits. In *Advances in Cryptology – EUROCRYPT 2013*. Springer Berlin Heidelberg, 336–352. http://dx.doi.org/10.1007/978-3-642-38348-9_21

[12] Sylvain Chatel, Christian Knabenhans, Apostolos Pyrgelis, Carmela Troncoso, and Jean-Pierre Hubaux. 2024. VERITAS: Plaintext Encoders for Practical Verifiable Homomorphic Encryption. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, U.S.A, October 14-18, 2024*. ACM. https://doi.org/10.1145/3658644.3670282

[13] Sylvain Chatel, Christian Mouchet, Ali Utkan Sahin, Apostolos Pyrgelis, Carmela Troncoso, and Jean-Pierre Hubaux. 2023. PELTA - Shielding Multiparty-FHE against Malicious Adversaries. In *ACM CCS 2023*, Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda (Eds.). ACM Press, 711–725. https://doi.org/10.1145/3576915.3623139

[14] Bhuvnesh Chaturvedi, Anirban Chakraborty, Ayantika Chatterjee, and Debdeep Mukhopadhyay. 2022. A Practical Full Key Recovery Attack on TFHE and FHEW by Inducing Decryption Errors. *Cryptology ePrint Archive* (2022). https://eprint.iacr.org/2022/1563

[15] Chia che Tsai, Donald E. Porter, and Mona Vij. 2017. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association, Santa Clara, CA, 645–658. https://www.usenix.org/conference/atc17/technical-sessions/presentation/tsai

[16] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. 2018. Labeled PSI from fully homomorphic encryption with malicious security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto Canada). ACM, New York, NY, USA. https://doi.org/10.1145/3243734.3243836

[17] Massimo Chenal and Qiang Tang. 2015. On Key Recovery Attacks Against Existing Somewhat Homomorphic Encryption Schemes. In *Progress in Cryptology - LATINCRYPT 2014*. Springer International Publishing, 239–258. http://dx.doi.org/10.1007/978-3-319-16295-9_13

[18] Jung Hee Cheon, Hyeongmin Choe, Alain Passelègue, Damien Stehlé, and Elias Suvanto. 2024. Attacks Against the INDCPA-D Security of Exact FHE Schemes. Cryptology ePrint Archive, Paper 2024/127. https://eprint.iacr.org/2024/127

[19] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Advances in Cryptology – ASIACRYPT 2017*. Springer International Publishing, 409–437. http://dx.doi.org/10.1007/978-3-319-70694-8_15

[20] Ilaria Chillotti, Nicolas Gama, and Louis Goubin. 2016. Attacking FHE-based applications by software fault injections. *Cryptology ePrint Archive* (2016). https://eprint.iacr.org/2016/1164

[21] Microsoft SEAL Contributors. 2022. Microsoft SEAL (release 4.0). https://github.com/Microsoft/SEAL. https://github.com/Microsoft/SEAL

[22] Anamaria Costache, Benjamin R. Curtis, Erin Hales, Sean Murphy, Tabitha Ogilvie, and Rachel Player. 2024. On the Precision Loss in Approximate Homomorphic Encryption. In *SAC 2023 (LNCS, Vol. 14201)*, Claude Carlet, Kalikinkar Mandal, and Vincent Rijmen (Eds.). Springer, Cham, 325–345. https://doi.org/10.1007/978-3-031-53368-6_16

[23] Anamaria Costache, Kim Laine, and Rachel Player. 2020. Evaluating the Effectiveness of Heuristic Worst-Case Noise Analysis in FHE. In *ESORICS 2020, Part II (LNCS, Vol. 12309)*, Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider (Eds.). Springer, Cham, 546–565. https://doi.org/10.1007/978-3-030-59013-0_27

[24] David Bruce Cousins, Yuriy Polyakov, Ahmad Al Badawi, Matthew French, Andrew Schmidt, Ajey Jacob, Benedict Reynwar, Kellie Canida, Akhilesh Jaiswal, Clynn Mathew, Homer Gamil, Negar Neda, Deepraj Soni, Michail Maniatakos, Brandon Reagen, Naifeng Zhang, Franz Franchetti, Patrick Brinich, Jeremy Johnson, Patrick Broderick, Mike Franusich, Bo Zhang, Zeming Cheng, and Massoud Pedram. 2023. TREBUCHET: Fully Homomorphic Encryption Accelerator for Deep Computation. (April 2023). arXiv:2304.05237 [cs.CR] http://arxiv.org/abs/2304.05237

[25] Rafael del Pino, Vadim Lyubashevsky, and Gregor Seiler. 2019. Short discrete log proofs for FHE and ring-LWE ciphertexts. In *Public-Key Cryptography – PKC 2019*. Springer International Publishing, Cham, 344–373. https://doi.org/10.1007/978-3-030-17253-4_12

[26] Keita Emura. 2021. On the Security of Keyed-Homomorphic PKE: Preventing Key Recovery Attacks and Ciphertext Validity Attacks. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E104.A, 1 (2021), 310–314. http://dx.doi.org/10.1587/transfun.2020EAL2039

[27] Keita Emura, Goichiro Hanaoka, Koji Nuida, Go Ohtake, Takahiro Matsuda, and Shota Yamada. 2018. Chosen ciphertext secure keyed-homomorphic public-key cryptosystems. *Des. Codes Cryptogr.* 86, 8 (Aug. 2018), 1623–1683. https://doi.org/10.1007/s10623-017-0417-6

[28] David Evans, Vladimir Kolesnikov, and Mike Rosulek. 2018. A Pragmatic Introduction to Secure Multi-Party Computation. *Foundations and Trends® in Privacy and Security* 2, 2-3 (2018), 70–246. https://doi.org/10.1561/3300000019

[29] Prastudy Fauzi, Martha Norberg Hovd, and Håvard Raddum. 2022. On the IND-CCA1 Security of FHE Schemes. *Cryptography* 6, 1 (2022). https://doi.org/10.3390/cryptography6010013

[30] Shufan Fei, Zheng Yan, Wenxiu Ding, and Haomeng Xie. 2021. Security Vulnerabilities of SGX and Countermeasures: A Survey. *ACM Comput. Surv.* 54, 6 (July 2021), 1–36. https://doi.org/10.1145/3456631

[31] Dario Fiore, Rosario Gennaro, and Valerio Pastro. 2014. Efficiently Verifiable Computation on Encrypted Data. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (Scottsdale, Arizona, USA) *(CCS '14)*. Association for Computing Machinery, New York, NY, USA, 844–855. https://doi.org/10.1145/2660267.2660366

[32] Dario Fiore, Anca Nitulescu, and David Pointcheval. 2020. Boosting Verifiable Computation on Encrypted Data. In *Public-Key Cryptography – PKC 2020*. Springer International Publishing, 124–154. http://dx.doi.org/10.1007/978-3-030-45388-6_5

[33] Antonio Merino Gallardo and Christian Knabenhans. 2023. circomlib-FHE. https://github.com/zkFHE/circomlib-fhe

[34] Chaya Ganesh, Anca Nitulescu, and Eduardo Soria-Vazquez. 2023. Rinocchio: SNARKs for Ring Arithmetic. *Journal of Cryptology* 36, 4 (Oct. 2023), 41. https://doi.org/10.1007/s00145-023-09481-3

[35] Robin Geelen, Michiel Van Beirendonck, Hilder V L Pereira, Brian Huffman, Tynan McAuley, Ben Selfridge, Daniel Wagner, Georgios Dimou, Ingrid Verbauwhede, Frederik Vercauteren, and David W Archer. 2022. BASALISC: Programmable asynchronous hardware accelerator for BGV fully Homomorphic Encryption. (May 2022). arXiv:2205.14017 [cs.CR] http://arxiv.org/abs/2205.14017

[36] Rosario Gennaro, Craig Gentry, and Bryan Parno. 2010. Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. In *Advances in Cryptology – CRYPTO 2010*. Springer Berlin Heidelberg, 465–482. http://dx.doi.org/10.1007/978-3-642-14623-7_25

[37] Rosario Gennaro, Michele Minelli, Anca Nitulescu, and Michele Orrù. [n. d.]. *Lattice-Based zk-SNARKs from Square Span Programs*. Technical Report.

[38] Rosario Gennaro and Daniel Wichs. 2013. Fully Homomorphic Message Authenticators. In *Advances in Cryptology - ASIACRYPT 2013*. Springer Berlin Heidelberg, 301–320. http://dx.doi.org/10.1007/978-3-642-42045-0_16

[39] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 48)*, Maria Florina Balcan and Kilian Q Weinberger (Eds.). PMLR, New York, New York, USA, 201–210. https://proceedings.mlr.press/v48/gilad-bachrach16.html

[40] Shruthi Gorantala, Rob Springer, Sean Purser-Haskell, William Lam, Royce Wilson, Asra Ali, Eric P Astor, Itai Zukerman, Sam Ruth, Christoph Dibak, Phillipp Schoppmann, Sasha Kulankhina, Alain Forget, David Marn, Cameron Tew, Rafael Misoczki, Bernat Guillen, Xinyu Ye, Dennis Kraft, Damien Desfontaines, Aishe Krishnamurthy, Miguel Guevara, Irippuge Milinda Perera, Yurii Sushko, and Bryant Gipson. 2021. A general purpose transpiler for fully homomorphic encryption. (June 2021). arXiv:2106.07893 [cs.CR] https://research.google/pubs/pub50428/

[41] Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments. In *Advances in Cryptology – EUROCRYPT 2016*. Springer Berlin Heidelberg, 305–326. http://dx.doi.org/10.1007/978-3-662-49896-5_11

[42] Seungwan Hong, Jai Hyun Park, Wonhee Cho, Hyeongmin Choe, and Jung Hee Cheon. 2022. Secure tumor classification by shallow neural network using homomorphic encryption. *BMC genomics* 23, 1 (9 April 2022), 284. https://doi.org/10.1186/s12864-022-08469-w

[43] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. {GAZELLE}: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium (USENIX Security 18)*. 1651–1669. https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar

[44] Marcel Keller, Valerio Pastro, and Dragos Rotaru. 2018. Overdrive: Making SPDZ Great Again. In *Advances in Cryptology – EUROCRYPT 2018*. Springer International Publishing, 158–189. https://doi.org/10.1007/978-3-319-78372-7_6

[45] Andrey Kim, Yuriy Polyakov, and Vincent Zucca. 2021. Revisiting Homomorphic Encryption Schemes for Finite Fields. In *ASIACRYPT 2021, Part III (LNCS, Vol. 13092)*, Mehdi Tibouchi and Huaxiong Wang (Eds.). Springer, Cham, 608–639. https://doi.org/10.1007/978-3-030-92078-4_21

[46] Christian Knabenhans. 2023. ringSNARK. https://github.com/zkFHE/ringSNARK

[47] Christian Knabenhans. 2024. zkOpenFHE. https://github.com/zkFHE/zkOpenFHE

[48] Christian Knabenhans, Alexander Viand, Antonio Merino-Gallardo, and Anwar Hithnawi. 2023. vFHE: Verifiable Fully Homomorphic Encryption. https://arxiv.org/abs/2301.07041. https://arxiv.org/abs/2301.07041 Extended version.

[49] Junzuo Lai, Robert H Deng, Changshe Ma, Kouichi Sakurai, and Jian Weng. 2016. CCA-Secure Keyed-Fully Homomorphic Encryption. In *Public-Key Cryptography – PKC 2016*. Springer Berlin Heidelberg, 70–98. http://dx.doi.org/10.1007/978-3-662-49384-7_4

[50] Kristin Lauter, Sreekanth Kannepalli, Kim Laine, and Radames Cruz Moreno. 2021. Password Monitor: Safeguarding passwords in Microsoft Edge. https://www.microsoft.com/en-us/research/blog/password-monitor-safeguarding-passwords-in-microsoft-edge/

[51] Baiyu Li and Daniele Micciancio. 2021. On the Security of Homomorphic Encryption on Approximate Numbers. In *Advances in Cryptology – EUROCRYPT 2021*. Springer International Publishing, 648–677. http://dx.doi.org/10.1007/978-3-030-77870-5_23

[52] Shimin Li, Xin Wang, and Rui Zhang. 2018. Privacy-Preserving Homomorphic MACs with Efficient Verification. In *Web Services – ICWS 2018*. Springer International Publishing, 100–115. http://dx.doi.org/10.1007/978-3-319-94289-6_7

[53] Zengpeng Li, Steven D Galbraith, and Chunguang Ma. 2016. Preventing Adaptive Key Recovery Attacks on the GSW Levelled Homomorphic Encryption Scheme. In *Provable Security*. Springer International Publishing, 373–383. http://dx.doi.org/10.1007/978-3-319-47422-9_22

[54] Benoit Libert. 2023. Vector Commitments With Proofs of Smallness: Short Range Proofs and More. Cryptology ePrint Archive, Paper 2023/800. https://eprint.iacr.org/2023/800

[55] Jake Loftus, Alexander May, Nigel P Smart, and Frederik Vercauteren. 2012. On CCA-Secure Somewhat Homomorphic Encryption. In *Selected Areas in Cryptography*. Springer Berlin Heidelberg, 55–72. http://dx.doi.org/10.1007/978-3-642-28496-0_4

[56] F Luo and K Wang. 2018. Verifiable decryption for fully homomorphic encryption. *Security: 21st International Conference, ISC 2018 …* (2018). https://link.springer.com/chapter/10.1007/978-3-319-99136-8_19

[57] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On Ideal Lattices and Learning with Errors over Rings. In *Advances in Cryptology – EUROCRYPT 2010*. Springer Berlin Heidelberg, Berlin, Heidelberg, Berlin, Germany, 1–23. https://doi.org/10.1007/978-3-642-13190-5_1

[58] Kit Murdock, David Oswald, Flavio D Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. 2020. Plundervolt: Software-based Fault Injection Attacks against Intel SGX. In *2020 IEEE Symposium on Security and Privacy (SP)* (San Francisco, CA, USA). IEEE, 1466–1482. http://dx.doi.org/10.1109/SP40000.2020.00057

[59] M Naor and M Yung. 1990. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the twenty-second annual ACM symposium on Theory of Computing* (Baltimore, Maryland, USA) *(STOC '90)*. Association for Computing Machinery, New York, NY, USA, 427–437. https://doi.org/10.1145/100216.100273

[60] D. Natarajan, A. Loveless, W. Dai, and R. Dreslinski. 2023. CHEX-MIX: Combining Homomorphic Encryption with Trusted Execution Environments for Oblivious Inference in the Cloud. (jul 2023), 73–91. https://doi.org/10.1109/EuroSP57164.2023.00014

[61] Ng and Chow. 2023. SoK: Cryptographic Neural-Network Computation. In *2023 IEEE Symposium on Security and Privacy (SP)*, Vol. 0. 497–514. http://dx.doi.org/10.1109/SP46215.2023.00198

[62] Alexander Nilsson, Pegah Nikbakht Bideh, and Joakim Brorsson. 2020. A Survey of Published Attacks on Intel SGX. (June 2020). arXiv:2006.13598 [cs.CR] http://arxiv.org/abs/2006.13598

[63] openenclave contributors. 2022. openenclave: SDK for developing enclaves. https://github.com/openenclave/openenclave Accessed: 2024-07-26.

[64] Oded Regev. 2009. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. *J. ACM* 56, 6 (Sept. 2009), 34:1–34:40. https://doi.org/10.1145/1568318.1568324

[65] Tom Rondeau. 2020. Data protection in virtual environments (DPRIVE). https://www.darpa.mil/program/data-protection-in-virtual-environments

[66] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Srinivas Devadas, Ronald Dreslinski, Christopher Peikert, and Daniel Sanchez. 2021. F1: A Fast and Programmable Accelerator for Fully Homomorphic Encryption. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece) *(MICRO '21)*. Association for Computing Machinery, New York, NY, USA, 238–252. https://doi.org/10.1145/3466752.3480070

[67] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Nathan Manohar, Nicholas Genise, Srinivas Devadas, Karim Eldefrawy, Chris Peikert, and Daniel Sanchez. 2022. CraterLake: a hardware accelerator for efficient unbounded computation on encrypted data. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 173–187. https://doi.org/10.1145/3470496.3527393

[68] Shingo Sato, Keita Emura, and Atsushi Takayasu. 2022. Keyed-Fully Homomorphic Encryption Without Indistinguishability Obfuscation. In *ACNS 22International Conference on Applied Cryptography and Network Security (LNCS, Vol. 13269)*, Giuseppe Ateniese and Daniele Venturi (Eds.). Springer, Cham, 3–23. https://doi.org/10.1007/978-3-031-09234-3_1

[69] Alexander Viand, Patrick Jattke, Miro Haller, and Anwar Hithnawi. 2023. HECO: Fully Homomorphic Encryption Compiler. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 4715–4732. https://www.usenix.org/conference/usenixsecurity23/presentation/viand

[70] Alexander Viand, Patrick Jattke, and Anwar Hithnawi. 2021. SoK: Fully Homomorphic Encryption Compilers. In *2021 IEEE Symposium on Security and Privacy (SP)*. ieeexplore.ieee.org, 1092–1108. http://dx.doi.org/10.1109/SP40001.2021.00068

[71] Biao Wang, Xueqing Wang, and Rui Xue. 2018. CCA1 secure FHE from PIO, revisited. *Cybersecurity* 1, 1 (Sept. 2018), 1–8. https://cybersecurity.springeropen.com/articles/10.1186/s42400-018-0013-8

[72] Biao Wang, Xueqing Wang, and Rui Xue. 2018. CCA1 secure FHE from PIO, revisited. *Cybersecurity* 1, 1 (25 Sept. 2018), 1–8. https://doi.org/10.1186/s42400-018-0013-8

[73] Xiaofeng Wang, Haixu Tang, Shuang Wang, Xiaoqian Jiang, Wenhao Wang, Diyue Bu, Lei Wang, Yicheng Jiang, and Chenghong Wang. 2018. iDASH secure genome analysis competition 2017. *BMC Med. Genomics* 11, Suppl 4 (Oct. 2018), 85. http://dx.doi.org/10.1186/s12920-018-0396-0

[74] Zama. 2022. Concrete: TFHE Compiler that converts python programs into FHE equivalent.

[75] Zhenfei Zhang, Thomas Plantard, and Willy Susilo. 2012. Reaction Attack on Outsourced Computing with Fully Homomorphic Encryption Schemes. In *Information Security and Cryptology - ICISC 2011*. Springer Berlin Heidelberg, 419–436. http://dx.doi.org/10.1007/978-3-642-31912-9_28