



USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

GlitchFHE: Attacking Fully Homomorphic Encryption Using Fault Injection

Lakshmi Likhitha Mankali and Mohammed Nabeel, *Tandon School of Engineering,
New York University*; Faiq Raees, *New York University*; Michail Maniatakos,
Ozgur Sinanoglu, and Johann Knechtel, *New York University Abu Dhabi*

<https://www.usenix.org/conference/usenixsecurity25/presentation/mankali>

**This paper is included in the Proceedings of the
34th USENIX Security Symposium.**

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Proceedings of the
34th USENIX Security Symposium is sponsored by USENIX.

GlitchFHE: Attacking Fully Homomorphic Encryption Using Fault Injection

Lakshmi Likhitha Mankali,¹ Mohammed Nabeel,¹ Faiq Raees,²
Michail Maniatakos,³ Ozgur Sinanoglu,³ Johann Knechtel³

¹ Tandon School of Engineering, New York University, USA <lm4344@nyu.edu, mtn2@nyu.edu>; both authors contributed equally.

² New York University, USA <fr872@nyu.edu>

³ New York University Abu Dhabi, UAE <mm6446@nyu.edu, os22@nyu.edu, jk176@nyu.edu>

Abstract

Fully homomorphic encryption (FHE) enables outsourced computation on encrypted data, thereby ensuring *confidentiality* by design. However, a critical vulnerability remains for FHE: cloud-based adversaries could tamper with the encrypted data itself, undermining the *integrity* of FHE applications. Fault-injection attacks (FIAs) are particularly concerning, as they allow attackers to directly manipulate the encrypted data during computation within FHE hardware.

Here, we present the first in-depth study of FIAs on FHE accelerators. We identify and overcome key challenges for such attacks, namely (i) understanding when and where to glitch encrypted data across different FHE applications and formats, (ii) limiting the number of required faults, and (iii) controlling the errors' impact on the final, decrypted output. We develop *GlitchFHE*, an analytical framework that guides toward such adversarial security assessment of FHE applications against FIAs. We run through a set of timely and relevant case studies, covering established applications like image processing, neural networks, and polynomial evaluation, for the prominent CKKS and BFV schemes, both at the analytical/software level and the physical/real-world level. Ultimately, we show that attacks on data integrity are a serious threat for FHE. Our work is equally relevant for real-world attackers and to educate FHE developers on this largely overlooked threat. We release *GlitchFHE* and our case studies at <https://doi.org/10.5281/zenodo.15615934>.

1 Introduction

In today's digital age, cloud-based services have become integral to modern computing, offering unparalleled scalability, flexibility, and accessibility for data processing and storage [1]. Such services are adopted by organizations in various sectors, including e-commerce platforms, social media, healthcare systems, and financial services, which all handle vast amounts of sensitive data [2–4]. However, as organizations increasingly rely on cloud infrastructure, concerns around data privacy and security have escalated [2].

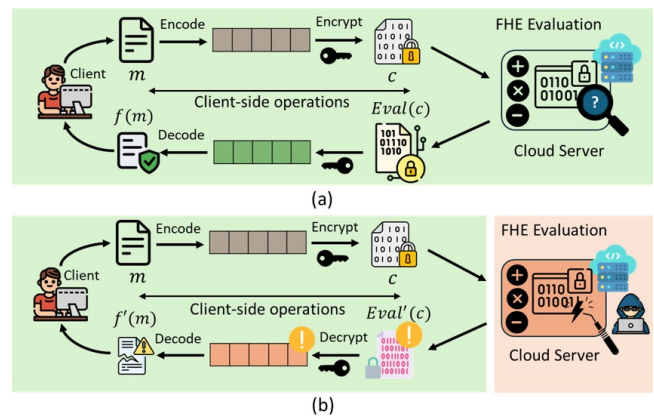


Figure 1: Data flow and threat model for FHE-based computation. (a) shows the conventional FHE flow that maintains *data confidentiality*. For a given encrypted text $c = Enc(m)$ from the client's end, $Eval(c)$ is performed on a cloud server and returned back to the client to decrypt and decode it. (b) shows threats on *data integrity*. While the client-side operations remain secure, a malicious entity at the server can manipulate FHE processing through mechanisms such as fault injection.

Established cryptography, while effective in securing data for transmission and storage, cannot protect data during processing in the cloud: there, data must be decrypted before the actual computation. Providing access to decrypted data in the cloud, however, can result in financial losses and leakage of confidential information such as bank account details, social security numbers, or healthcare data. For instance, the *IBM 2023 Cost of Data Breach Study* [5] highlights some implications of prominent data breaches, with an average cost of approximately \$4.45 million per breach, and a significant 82% of breaches involving data processed in the cloud.

Fully Homomorphic Encryption (FHE). FHE has emerged as a promising solution to the above-mentioned challenges. It ensures the *confidentiality* of data during computation [6], by enabling users to securely outsource computations on encrypted data. More specifically, FHE facilitates the ex-

execution of arbitrary computations over encrypted data while producing the same encrypted results as if the same computations were performed on plaintext data [6–9]. This core capability of FHE eliminates the need for decryption of data, eliminating traditional data risks (Fig. 1(a)). Here, some plaintext m is encoded and encrypted into ciphertext $c = \text{Enc}(m)$ at the client side. This encrypted data c is sent to a service provider where it undergoes computationally intensive processing operations, realized through homomorphic evaluation on ciphertexts, represented by $\text{Eval}(c)$. Importantly, the service provider cannot read the plaintexts, only the ciphertexts. Sent back to the client, the processed ciphertext is decrypted and decoded to retrieve the plaintext data of the result, through $f(m) = \text{Dec}(\text{Eval}(f, c))$. This process ensures *data confidentiality* end-to-end.

FHE is proposed for several privacy-sensitive real-world applications, such as genomics [10, 11], healthcare [12, 13], and national security [14]. FHE has attracted widespread attention from government, commercial entities, and academia. For instance, DARPA is investing considerably in R&D of dedicated hardware accelerators for FHE through programs such as Programming Computation on Encrypted Data (PROCEED) [15] and Data Protection in Virtual Environments (DPRIVE) [16], with budgets up to 14 million dollars. Besides, companies such as Microsoft, IBM, and Intel have also invested in and/or developed FHE-based applications and hardware accelerators [17–20].

Although FHE holds great promise for confidentiality, there is a risk for *data integrity*, i.e., the accuracy and correctness of data for computations performed by outsourced third parties. This is because, once clients send their encrypted data (aka ciphertexts) to servers, clients do not have control over that data as such anymore. A malicious entity within the server environment could manipulate (i) the encrypted data itself and/or (ii) the FHE operations, which would both undermine the trustworthiness of FHE applications (Fig. 1(b)).

Research Gap. Prior art for adversarial security assessment of FHE schemes has only focused on confidentiality threats, e.g., to recover secret keys [21–24]. Further, prior art for security assessment is primarily limited to software implementations of FHE schemes; see Sec. 6.1 for more details. Thus, despite the potential severity of the threat, data integrity of FHE schemes has been largely overlooked so far.¹

Scope and Contributions. In this work, we address this gap. For the first time in the literature, we explore the risks for data integrity associated with FHE computation handled by outsourced service providers hosting hardware accelerators. Specifically, with direct access to the hardware for server-side adversaries, physical attacks like fault-injection attacks

(FIAs) pose a potent threat. We proactively play the role of attackers with FIA capabilities, a well-established adversarial technique that involves inducing transient malfunctions or “glitches” in the hardware to manipulate data.

We show how such FIAs on FHE accelerators can corrupt the encrypted data, resulting in incorrect computation outputs. However, we also show that such attacks are not straightforward, due to the complexity of ciphertext formats and various FHE operations performed on ciphertexts.

We overcome related technical challenges while developing an analytical framework for assessment of FIAs on FHE schemes, called *GlitchFHE*. More specifically, *GlitchFHE* enables security researchers to (i) cause reasonable, error-limited corruption in the output, (ii) glitch encrypted data in different formats, (iii) require only a limited number of actual faults, and (iv) assess application-specific integrated circuit (ASIC)-based FHE accelerators in a systematic manner.

Note that *GlitchFHE* is agnostic, i.e., it can be used for both adversarial and defense efforts. For adversarial efforts, *GlitchFHE* guides toward meaningful FIAs on FHE accelerators such that data corruption remains within limits. For defense efforts, *GlitchFHE* informs developers about shortcomings for FHE security in general and about the degree of data corruption to expect by successful FIAs in particular.

Our contributions can be summarized as follows.

1. We introduce *GlitchFHE*, a novel framework that enables proactive, adversarial-driven security assessment of various FHE applications against integrity attacks on encrypted data. To the best of our knowledge, despite the severity of this threat, *GlitchFHE* is a first for such efforts.
2. We showcase the applicability and efficacy of *GlitchFHE*. We address various technical challenges outlined above. We examine the prominent CKKS and BFV FHE schemes, while *GlitchFHE* is also extendible for other schemes.
3. Through a range of case studies, we demonstrate end-to-end security assessment of representative FHE applications, including the prominent machine learning (ML)-centric *CryptoNets* accelerators [25]. Our assessment is guided by analytical insights obtained from *GlitchFHE* and includes actual physical attacks using an electromagnetic FIA setup on *CoFHEE*, a state-of-the-art ASIC-based FHE accelerator [26].
4. We release the *GlitchFHE* methodology and our case studies to the public at <https://doi.org/10.5281/zenodo.15615934>.

2 Background and Motivation

2.1 Overview for FHE

Here, we provide a brief overview on state-of-the-art FHE schemes and their key working principles. Importantly, we

¹We acknowledge recent efforts toward *verifiable* FHE; see also Sec. 6.4. However, such efforts are (i) very costly to implement and (ii) difficult to generalize, i.e., they do not cover/protect all the different stages of various FHE schemes. Thus, even for verifiable FHE, end-to-end integrity remains missing, opening up severe attack surfaces as we showcase in this work.

also relate this review to vulnerabilities to FIAs based on how errors are handled within FHE schemes.

Inherent Errors. Popular FHE schemes such as BFV [7], BGV [8], CKKS [9], and TFHE/CGGI [27] are all based on the so-called (*ring*) *learning-with-error ((R)LWE) problem*. In (R)LWE-based FHE schemes, to ensure security by means of making unauthorized reversing of encryption difficult, a small noise is introduced when generating a public key or encrypting a plaintext. Thus, all the ciphertexts generated within these schemes contain some errors by design, whereas the error grows throughout the FHE computation procedure; the latter is also known as *homomorphic evaluation*.

FHE schemes require some measures for keeping these by-construction errors within bounds, such that the integrity of the processed data is maintained. These measures are outlined next. Importantly, however, these measures and their applications are based on specific and static assumptions for the degree of errors, without consideration of any unexpected and additional errors. In other words, the error-handling measures integrated into FHE schemes cannot, per se, protect against additional error induced by FIAs.

Measures for Error Management. FHE schemes generally contain a basic component which is referred to as *some-what homomorphic encryption (SHE)*. SHE supports the evaluation of multiple (yet few) consecutive operations on ciphertexts. An important parameter here is *multiplication level* or *multiplicative depth*, which defines the maximum number of sequential multiplication operations that can be supported during FHE evaluation such that the decryption of the final result remains correct. SHE allows for the related growth of the ciphertext sizes through a technique called *relinearization*, such that the error accumulated after conducting all the operations is still within limits for a correct decryption. Note that accumulated errors can also be removed intermittently as needed, by a procedure called *bootstrapping*. However, bootstrapping constitutes a major performance bottleneck [28]; thus, it is utilized conservatively and differently throughout various FHE schemes, if at all.

Categorization and General Assessment. Popular FHE schemes are discussed next. BFV [7], BGV [8], and CKKS [9] are all leveled schemes, i.e., to limit the related performance impact, they perform bootstrapping only after a few rounds/levels of multiplicative operations. Furthermore, these schemes all operate at the word-level. BFV and BGV support integer format, whereas CKKS supports real numbers.

BGV and BFV are both suitable for working with finite fields in modular exact/integer arithmetic. Furthermore, as they both support *batching*, a process for efficient packing of multiple plaintexts into a single ciphertext, the readily support single-instruction-multiple-data (SIMD) operations to perform computation over sets of integers more efficiently. Thus, these schemes are well-suited for manipulation of ring objects in the form of polynomials.

Table 1: Common FHE parameters and description.

Parameter	Description
N	$N - 1$ is the plaintext polynomial degree.
Q	Ciphertext polynomial modulus.
Δ	CKKS scaling factor. Controls the precision of the fractional part of real/complex plaintext.
L	Maximum multiplication level.
ℓ	Current multiplication level. $0 \leq \ell \leq L$.
Q'_ℓ	Ciphertext polynomial modulus at level ℓ in its non-RNS form; $Q_\ell = q_0 \cdot \Delta^\ell$.
Q_ℓ	Ciphertext polynomial modulus at level ℓ in RNS-form; $Q_\ell = \prod_{i=0}^l q_i$.
q_i	i th RNS prime base of Q_ℓ .
M_i	Q_ℓ / q_i
p	Extra RNS base added to expand the polynomial modulus from Q to $Q \cdot p$.
μ	Ciphertext/Message vector.
$\mathbf{m}$	Plaintext polynomial.
$\mathbf{c}$	Ciphertext polynomial pair: $(\mathbf{a}, \mathbf{b})$.
$\mathbf{s}$	Secret key polynomial.
$\mathbf{e}$	Error/Noise present in the ciphertext.
$\mathbf{x}$	Modulo Q_ℓ polynomial with degree $N - 1$.
$[\mathbf{x}]_q$	$\mathbf{x} \bmod q$.

As indicated, CKKS supports real-number arithmetic. It also supports batching with more slots, i.e., can pack more data for SIMD operations than the BFV and BGV schemes. Thus, CKKS is currently the most efficient scheme for performing approximate homomorphic computations over real and complex numbers, which is essential for any real-world problems like image processing.

2.2 Key FHE Concepts, Explained on CKKS

Next, we provide more details on the Cheon-Kim-Kim-Song (CKKS) scheme [9]. Parameters used throughout are summarized in Table 1. Within this review of CKKS, we also discuss important concepts that are applicable to FHE schemes in general: the *residue number system (RNS)*, the (*inverse*) *number theoretic transformation ((i)NTT)*, and *rescaling*. These concepts will become important for FIAs in general and for *GlitchFHE* in particular.

CKKS Overview. Figure 2 shows the computation/evaluation steps involved in CKKS. A message μ , i.e., a vector of values on which certain computations need to be performed to evaluate $f(\mu)$, is first encoded into a plaintext polynomial $\mathbf{m}$ and then multiplied with a scaling factor Δ . Next, the polynomial is encrypted, using a secret key $\mathbf{s}$, into a pair of ciphertext polynomials $\mathbf{c} = (\mathbf{a}, \mathbf{b})$. Then, $f(\mathbf{a}, \mathbf{b})$ is evaluated, which contains a series of homomorphic operations that realize the actual, application-specific computation. Finally, the evaluated ciphertext is decrypted and decoded, providing $f(m)$.

CKKS Message Encoding and Decoding. For CKKS, the plaintext is a set of size $N/2$, with each element being a complex number $\mathbb{C}$. This input vector is expanded to μ of size N , by copying the so-called conjugate. Importantly, adding

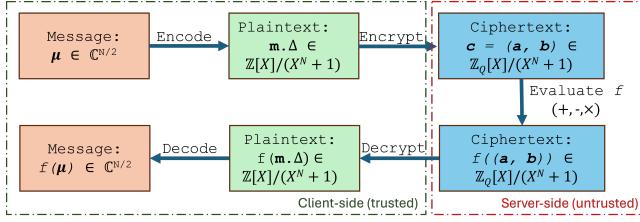


Figure 2: CKKS main stages/operations and data flow.

the conjugate ensures that the encoded polynomial does not retain any imaginary component. Next, the vector μ is treated as a set of evaluation points, x_j , of the polynomial $\mathbf{m} \in \mathbb{R}[X]$, such that $\mu = \{\mathbf{m}(x_j)\}$. Note that all the encoded values are represented by the coefficients of the polynomial $\mathbf{m}$.

CKKS Scaling. The coefficients of the encoded polynomial $\mathbf{m} \in \mathbb{R}$ have to be converted to an integer to be used as an RLWE element. This is done by multiplying them with a *scaling factor* Δ . In other words, multiplication with Δ gets rid of the decimal points in the data, with some precision loss. Typically, Δ is chosen as a power of two, and the higher the power, the better the precision. That is, post-decryption but before decoding, the decrypted value is divided again by Δ ; this maintains a precision of $\frac{1}{\Delta}$. More details on the division are also provided further below.

CKKS Ciphertext Format. Here, a ciphertext $\mathbf{c}$ is a pair of polynomials $(\mathbf{a}, \mathbf{b}) \in R_{Q_\ell} = \mathbb{Z}_{Q_\ell}[X]/(X^N + 1)$, i.e., a polynomial of at most $N - 1$ degree with coefficients $\in \mathbb{Z}_{Q_\ell} \mid Q_\ell = \Delta^\ell \cdot q_0$. Note that Q_ℓ is the modulus of the ciphertext at multiplication level ℓ , and Δ is the scaling factor introduced above.

Now, after FHE evaluation by operation f , a ciphertext $\mathbf{c}' = (\mathbf{a}', \mathbf{b}')$ is called an encryption of the plaintext $f(\mathbf{m})$, if decryption of $\mathbf{c}'$, $\text{Dec}(\mathbf{c}') = \mathbf{b}' + \mathbf{a}' \cdot \mathbf{s} = \Delta \cdot f(\mathbf{m}) + \mathbf{e}' \approx \Delta \cdot \mathbf{m}$, where $\mathbf{e}'$ is the accumulated error after FHE evaluation. In other words, after evaluation, $\|\mathbf{e}'\|$ should be low enough so that the decrypted value is approximately equal to $\Delta \cdot f(\mathbf{m})$. Note that $\mathbf{a}$ can be considered as a polynomial randomly sampled from the ring R_{Q_ℓ} and $\mathbf{b} = -\mathbf{a} \cdot \mathbf{s} + \Delta \cdot \mathbf{m} + \mathbf{e}$, where the error polynomial $\mathbf{e}$ is from a discrete Gaussian distribution [29] and secret key $\mathbf{s}$ is uniformly sampled from $\{-1, 0, 1\}^n$. During the FHE evaluation, whenever the ciphertext undergoes operations like addition/subtraction, the error will increase.

RNS. For any FHE scheme, the ciphertext modulus Q can be chosen as a product of small, pairwise coprime moduli $q_0, q_1, \dots, q_l$. Then, one can make use of the *Chinese remainder theorem (CRT)* [30], which offers some *ring isomorphism* from $\mathbb{Z}_Q$ to $\prod \mathbb{Z}_{q_i}$. In other words, applying the CRT over $\mathbb{Z}_Q$ will result in a non-positional number system, called RNS, where a large integer in $\mathbb{Z}_Q$ is mapped to small residues. Thus, any arithmetic in $\mathbb{Z}_Q$ can be split into l independent operations on the residues $\mathbb{Z}_{q_i}$.

Note that the ciphertext modulus Q can be as large as a few thousand bits for many FHE schemes. Thus, for better

performance, it is suggested to use RNS at the coefficient level, i.e., to split those into smaller words that are matching the machine-word size [30]. Then, $\mathbf{x}$ can be reconstructed from its bases as follows:

$$\mathbf{x} = \left[\sum [\mathbf{x}]_{q_i} \cdot [M_i^{-1}]_{q_i} \cdot M_i \right]_Q \quad (1)$$

where $M_i = Q/q_i$. This equation is important to determine the impact of data corruption for an RNS polynomial coefficient on the actual ciphertext polynomial and eventually on the input plaintext. Thus, for *GlitchFHE*, using this equation, we will derive the amount of corruption to be done at the coefficient level through FIAs such that meaningful adversarial effects can be realized on the input plaintext.

NTT. As indicated, polynomial multiplication is at the heart of any FHE scheme. When done in a naive way, such multiplication holds a complexity of $O(N^2)$. With NTT, however, this can be reduced to $O(N \log N)$.

NTT is a generalized form of discrete Fourier transformation (DFT) which operates on an integer modulo of a prime number q . Similar to the complex roots of unity used in DFT for a complex number field, NTT employs the primitive N^{th} -root of unity w , so that $w^N = 1 \bmod q$, where N is the polynomial degree. Note that the powers of primitive N^{th} of unity are called the *twiddle factors*. As with DFT, fast Fourier transformation (FFT) can be used to perform this transformation with complexity $O(N \log N)$.

iNTT. This inverse operation maps data from the NTT domain back to the polynomial domain. For an input vector $X = [X_0, X_1, \dots, X_{N-1}]$ in the NTT domain, the iNTT produces an output vector $x = [x_0, x_1, \dots, x_{N-1}]$ in the polynomial domain as follows:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot \omega^{-k \cdot n} \bmod p, \quad \text{for } n = 0, 1, \dots, N-1 \quad (2)$$

where ω is a primitive N -th root of unity modulo p , and p is a prime modulus. NTT restricts the coefficient modulus p to be congruent to $1 \bmod 2N$.

Scope for FIAs, Part I. With NTT, the computational expensive polynomial multiplication becomes an element-wise multiplication. Thus, in many FHE schemes, polynomials at rest are stored in NTT form and converted to their coefficient form only when an operation requires it. Later, we discuss the difficulty of FIAs when ciphertext polynomials are in NTT form and why it is preferred to run FIAs on the coefficient form. Accordingly, an understanding of when there is switching from the NTT form to the coefficient form during FHE evaluation is important for FIAs. One prominent operation toward that end is *rescaling*, which we discuss next.

CKKS Rescaling. Recall that both the coefficients $\mathbf{a}$ and $\mathbf{b}$ are multiplied with Δ as part of encoding. Thus, post-multiplication of some ciphertexts, the result will include a component with the square of the scaling factor. If not handled, this component can quickly lead to overflows after just

a few multiplications, as the scale grows exponentially. To handle this, CKKS employs a process called rescaling.

For rescaling, the ciphertext's polynomial pairs are divided by the scaling factor Δ and then rounded. Since rounding cannot be performed in RNS representation, the value of each q_i is set close to Δ ; rescaling can be approximated to shrinking the modulus from Q_ℓ to $Q_{\ell-1}$. As rescaling is done after each and every multiplication, it also dictates the starting size of the modulus, based on the maximum level L to be $q_0 \cdot \Delta^L \approx \prod_i^L q_i$ [31]. Next, we provide pseudocode for rescaling.

```
def rescale([x]qi, qi, ℓ):
    result = []
    for j in range(0, ℓ):
        z = NTT([invNTT([x]qℓ)]qj)
        value = [qℓ-1([x]qj - z)]qj
        result.append(value)
    return (result)
```

Scope for FIAs, Part II. As can be seen in the above pseudocode, rescaling requires an exit from the NTT domain for the last tower x_{q_ℓ} , which can be exploited for FIAs.

In general, conversion from the NTT domain to the coefficient domain is achieved by *modulus switching*, where the underlying ciphertext modulus is switched to a different modulus. Thus, rescaling can be seen as switching from modulus Q_ℓ to $Q_{\ell-1}$, but it also involves scaling the ciphertext coefficients by q_ℓ^{-1} . Importantly, modulus switching is utilized in all the RLWE schemes. When performing computations in RNS, modulus switching means switching the RNS basis, i.e., convert $[x]_Q$ from its residue modulo $Q = q_0 \cdots q_{k-1}$ to $[x]_Q \bmod P$ for some $P = p_0 \cdots p_{k'-1}$. This is achieved by a technique called *fast base extension (FBE)* as follows:

$$FBE(x, Q, P) = \sum_{i=1}^k \left[[x]_{q_i} \left(\frac{Q}{q_i} \right)^{-1} \right] \frac{Q}{q_i} \bmod p_j \quad (3)$$

where the value of $[x]_{q_i}$ should be in the coefficient domain. As indicated, the widely used modulus-switching operations are of interest for FIAs in general. Without loss of generality, we exploit them for FIAs on BFV.

2.3 Hardware Accelerators for FHE

Various accelerators have been proposed to improve the performance of FHE computation. At first, the trend was to make use of features provided by general-purpose hardware, e.g., Intel introduced the Homomorphic Encryption Acceleration Library (HEXL) in 2021, utilizing AVX-512 instructions to speed up NTT operations [32]. Additionally, GPU implementations using CUDA programs have been employed by projects like *100×* [33], to accelerate CKKS. Despite offering significant acceleration for FHE schemes, they were still not enough to be practical for real-world applications. Next, researchers began to explore application-specific hardware. Many different approaches have been proposed, consequently leading to a wide range for state-of-the-art FHE acceleration

that are based on joint and careful orchestration of software, GPU, FPGA, and/or ASIC components. Note that most works are only simulator-based and/or they are not open-sourced, which significantly limits their usability for advancing further research and development. For more details, please refer to a recent survey paper by Zhang et al. [34].

As we utilize this in our case studies, we briefly discuss *CoFHEE* [26, 35], an open-sourced and silicon-proven hardware accelerator for low-level polynomial operations targeted for faster FHE evaluation. CoFHEE includes implementations of essential polynomial operations such as addition, subtraction, Hadamard product, and NTT, which are foundational to most higher-level FHE primitives. CoFHEE supports polynomial degrees up to 2^{13} with a coefficient modulus size of up to 128 bits. Considering the amount of internal memory CoFHEE provides, it can perform rescaling of ciphertext polynomials with up to four RNS towers. CoFHEE can be interfaced with any computer as external host and provides an API for programming [35].

2.4 Fault-Injection Attacks

FIAs are a well-established and proven threat vector. The objective is to insert transient or permanent faults into the hardware in order to modify the intended functionality of a system, e.g., to leak critical assets such as secret keys.

FIAs can be categorized based on their approach: (i) direct, semi-invasive/invasive fault injection, e.g., by laser light [36] or electromagnetic waves [37–39] versus (ii) indirect, e.g., by repetitive writing to particular memory locations [40] or by misuse of voltage and frequency features [41], etc. For a more comprehensive review of FIAs, please refer to [42].

Adversaries typically need to tune their efforts to achieve successful attacks: fault injection has to be carefully controlled to avoid glitching larger parts or even the whole system. Otherwise, the system might be “knocked out” where the actual attacks (e.g., leaking sensitive data over time) cannot proceed anymore. Even without such failures, careful control over fault injection is required to limit the induced errors to reasonable ranges. This is also a key challenge in our work.

3 Exploratory Study

A naive approach for FIAs on FHE could involve randomly injecting faults into the encrypted data's operation. However, such an approach could be easy to discern by the user/client, as the resulting corruption in outputs tends to be substantial. For instance, when randomly attacking a simple polynomial computation for CKKS in NTT domain, we observe that the mean square error (MSE) between expected and corrupted outputs was on average 3.74225×10^{14} , which is excessive. Note that the reason for this error is discussed later on.

To further illustrate this challenge in a practical manner, we introduce random faults into the encrypted data in NTT



Figure 3: Sobel filter implementation using FHE. Note that (a) represents the input, (b) represents the golden output without FIAs and corruption of encrypted data, (c) represents the output for randomized fault injection, whereas (d) represents the output for `GlitchFHE`-guided fault injection.

domain of a CKKS-based Sobel filter (i.e., a standard image-processing application used for edge detection). As shown in Fig. 3(c), with randomized fault injection, the resulting image clearly lacks patterns that would correspond to any meaningful edge detection, whereas a more careful fault injection guided by our proposed `GlitchFHE` framework could well pass as meaningful output for edge detection, as shown in Fig. 3(d), but is still sufficiently different/corrupted over the golden output shown in Fig. 3(b).

In short, from the attacker’s perspective, it is essential to devise a refined and targeted approach for fault injection to avoid excessive data corruption. In general, excessive data corruption refers to cases where decrypted outputs are invalid. For the given example of image processing, this could be defined as any decrypted pixel value falling outside $[0, 255]$ considering 8-bit pixel encoding.

4 Methodology

4.1 Threat Model

Attacker’s Location. As shown in Fig. 1, we consider an attacker located within untrusted cloud servers that offer FHE computations on encrypted data as a service. While FHE ensures the confidentiality of sensitive data during such outsourced computation, it does not guarantee the integrity of the encrypted data during processing; also recall Footnote 1.

Attacker’s Resources. We assume the following:

1. The attacker has access to the hardware performing FHE computations enabling direct interaction with the components handling the encrypted data.
2. The attacker knows the encryption technique and parameters of the encryption. This includes parameters such as the ciphertext polynomial modulus Q , RNS-based ciphertext moduli q_i , scaling factor Δ , and any additional parameter—except the secret key.

Attacker’s Capabilities. We assume that the attacker can monitor the operations on the FHE accelerator and inject faults into the hardware as outlined in Sec. 2.4. We consider the attacker to inject transient stuck-at faults, temporarily forcing some bits in the hardware to a fixed logic state (0 or 1),

thereby disrupting computations and potentially causing incorrect intermediate values. An attacker may introduce single or multiple faults, either simultaneously or sequentially. Furthermore, the attacker can target specific computational stages of the homomorphic operations to perform FIAs. These capabilities are in line with the literature for FIAs [42].

Attacker’s Goal. The attacker aims to disrupt critical applications, such as privacy-preserving ML in healthcare. The ultimate goal is not to steal sensitive information but to cause misclassification or corruption of sensitive data, thereby compromising the accuracy and reliability of sensitive applications through violation of data integrity.

4.2 General Problem Formulation

Let $Eval(f, c)$ represent the homomorphic evaluation of a function f on c . By introducing some fault δ , the adversary modifies the ciphertext to $c' = c + \delta$, where c' is the faulty ciphertext. Note that the fault δ can be modeled as bit-flips in binary form, expressed as $c = \sum_{i=0}^{w-1} b_i \cdot 2^i$, where $b_i \in \{0, 1\}$ denotes the i -th bit and w is the bit-width. A bit-flip at position i alters c by $\pm 2^i$, and a series of bit-flips corresponds to $\delta = \sum_{i \in F} \pm 2^i$, where F is the set of positions where flips occur.

The faulty ciphertext propagates through the computation, resulting in a perturbed output $Eval(f, c') = Eval(f, c + \delta)$. As indicated, the objective of the adversary is to introduce faults such that $Eval(f, c') \neq Eval(f, c)$, while ensuring that the effect is subtle and likely goes unnoticed in the final result.

There are important considerations here as follows. First, how to obtain some subtle glitch depends entirely on the FHE specifics; details are provided throughout the remainder of this section. Second, what constitutes as a glitch that is “subtle enough” and what not depends on both the adversary’s target and on the client’s understanding of nominal computation results. For the example of image processing in Sec. 3, this could be defined as any decrypted pixel value remaining within the valid range of $[0, 255]$ for the 8-bit encoding. Importantly, exploring the latter trade-off is out of scope for this work. Here, we focus on the foundation of this matter, namely how to successfully devise and inject faults such that specific error rates are achieved for the final computation results.

4.3 Inserting Faults in NTT Domain

Challenge ①: Recall that the operations on FHE accelerators are generally performed in NTT domain, as NTT-based computation is more efficient. Consequently, the encrypted data resides most of the time in the NTT domain. However, corrupting encrypted data in the NTT domain is not straightforward, as demonstrated in Sec. 3. This is due to the fact that, during decryption, the data is transformed back to the polynomial domain by the iNTT operation.² Thus, there is a

²While the iNTT is a linear transformation, the relationship between the NTT and polynomial domains is not direct in a way that would allow

need for a strategic approach to induce faults to the elements in NTT domain such that the effect for the outputs, after iNTT operation, remains controllable and reasonable.

Solution ①: To address this challenge, we perform a mathematical analysis for the propagation of faults during the iNTT operation. First, recall the definition of the iNTT operation in Equation 2, where ω is a primitive N -th root of unity modulo p , and N is the transformation size. This can be represented as $\mathbf{x} = \mathbf{M}_{\text{iNTT}} \cdot \mathbf{X}$, where $\mathbf{M}_{\text{iNTT}}$ is the iNTT matrix:

$$\mathbf{M}_{\text{iNTT}} = \frac{1}{N} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \dots & \omega^{-(N-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \dots & \omega^{-2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(N-1)} & \omega^{-2(N-1)} & \dots & \omega^{-(N-1)^2} \end{bmatrix} \quad (4)$$

The structure of this matrix reveals that the sum of each row, except for the first row, is zero; see also Appendix B. This means that, for any input vector $\mathbf{X}$ with identical values for each element (e.g., $X_k = \text{const}$ for all k), which is multiplied by the iNTT matrix, the output becomes:

$$x_0 = \frac{1}{N} \sum_{k=0}^{N-1} X_k, \quad x_n = 0 \text{ for } n > 0. \quad (5)$$

Now, consider a corrupted vector $\mathbf{X}_{\text{corr}} = \mathbf{X}_{\text{orig}} + \mathbf{X}_{\text{err}}$, where $\mathbf{X}_{\text{orig}}$ represents the original input and $\mathbf{X}_{\text{err}}$ is the induced error. By linearity of iNTT, we have $\text{iNTT}(\mathbf{X}_{\text{corr}}) = \text{iNTT}(\mathbf{X}_{\text{orig}}) + \text{iNTT}(\mathbf{X}_{\text{err}})$. If $\mathbf{X}_{\text{err}}$ is such that all its elements are identical, the iNTT transformation yields:

$$x_{\text{err},0} = \frac{1}{N} \sum_{k=0}^{N-1} X_{\text{err},k}, \quad x_{\text{err},n} = 0 \text{ for } n > 0. \quad (6)$$

In other words, a uniform error causes a controllable corruption of the iNTT output, specifically affecting only the first component $x_{\text{err},0}$ in $\mathbf{X}_{\text{err}}$ while all other components $x_{\text{err},n}$ for $n > 0$ remain as is.

To visualize this insight, we conduct an empirical study for $N = 4$ and show the results in Fig. 4. Confirming the mathematics above, one can see that only the input pattern ‘1111’ carries over the intended error, and also only for the first output, whereas all other patterns cause errors that (a) differ in location/output and (b) scale from the intended error.

This fact suggests that, to minimize the number of affected outputs, N elements in the original input X_{orig} must be subjected to an error. Thus, to induce an error e to the first output of iNTT transformation, the error e has to be induced to all the encrypted inputs in NTT domain.

simple and controlled error propagation. This is because the iNTT outputs are linearly dependent on all input values in the NTT domain, with each output element being a weighted sum of the inputs, as shown in Equation 2. This dependency means that any modification of the data in the NTT domain will affect all of the output values after decryption, as even small errors in the NTT domain can propagate and influence the entire sequence of decrypted values, as we also show in this subsection.

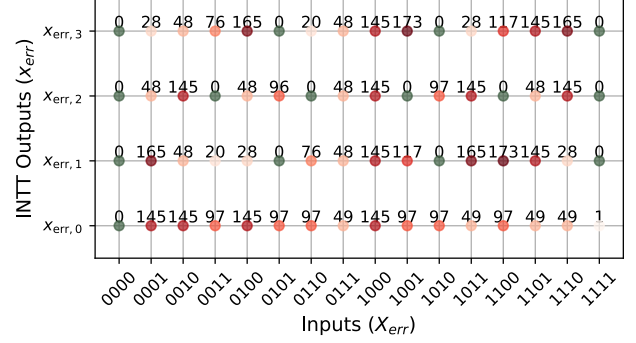


Figure 4: Effect of inducing a constant error, of exemplary value $X_{\text{err},k} = 1$, for all possible input patterns (for $N = 4$) during iNTT transformation. Note that the patterns are short form for the vector $\mathbf{X}_{\text{err}}$. Data points show the absolute errors over nominal iNTT outputs and are also color-coded with respect to the degree of corruption.

Table 2: Non-NTT operations targeted by GlitchFHE.

FHE Scheme	Non-NTT Operations
CKKS	Rescaling, Modulus Switching
BFV	Modulus Switching

4.4 Inserting Faults in Non-NTT Domain

Challenge ②. Note that Solution ① offers very limited control over the error propagation and necessitates a substantial number of simultaneous bit-flip faults. For example, in CKKS-based encrypted data with $N = 4096$, following our analysis above, approximately 10,000 concurrent bit-flips were required to produce a controllable error of $e = 1$ in the intermediate iNTT output of homomorphic operations. Naturally, this would pose significant practical challenges for any real-world FIA setup. Thus, a more efficient approach that minimizes the number of faults is needed.

Solution ②. Considering the fundamental challenges we discussed for controlled error propagation in the NTT/iNTT domain, targeting encrypted data in the non-NTT domain seems warranted. The FHE operations listed in Table 2 include those that partially process ciphertexts in the non-NTT domain for popular FHE schemes. It is also important to note that, while verifiable FHE schemes can safeguard the main arithmetic operations for FHE, they do not apply protection to these very operations; more details are discussed in Sec. 6.4. Therefore, our focus is directed towards these non-NTT domain operations for two reasons: to strive for controllable fault injection and to bypass recent verification efforts.

4.5 Optimizing Faults for RNS Ciphertexts

Challenge ③. The targeted operations listed in Table 2 convert only a single ciphertext polynomial into the non-NTT domain, namely that corresponding to the last RNS modulus.

Consequently, errors can only be introduced into this polynomial. However, doing so can led to excessive corruption in the output. As indicated in Sec. 2, for all RNS-based FHE schemes, ciphertexts are decomposed into different moduli using the CRT. Consequently, recall that, during decryption, the output is reconstructed using Equation 1. This process can contribute to large corruption as follows.

Let $q_1, \dots, q_n$ be a set of primes, and define $Q := \prod_{i \leq n} q_i$ and $M_i := \frac{M}{q_i}$. Then, given some integer inputs $a_1, \dots, a_n$, a solution to the following equation system exists: $x = a_1 \bmod q_1, \dots, x = a_n \bmod q_n$. In fact, the solution is unique: $x = \sum_{i \leq n} a_i M_i M_i^{-1} \bmod Q$, where $M_i M_i^{-1} = 1 \bmod q_i$.

Let x be the reference/golden solution and x' be the solution obtained after introducing an error somewhere in the inputs. Now, if we introduce the error in a_n , i.e., the coefficient corresponding to last modulus q_n , then

$$x - x' = \sum_{i \leq n} a_i M_i M_i^{-1} - \sum_{i \leq n} a_i M_i M_i^{-1} - e M_n M_n^{-1} \quad (7)$$

$$= e M_n M_n^{-1} \bmod Q. \quad (8)$$

This analysis reveals that the error becomes multiplied by $M_n M_n^{-1} \bmod Q$ in the reconstructed output, contributing to its possibly excessive impact. Indeed, we have confirmed this by injecting random errors for a simple polynomial computation for CKKS, where we observed MSE values up to 10^{14} , which would be discernible for the client. This challenge underscores the need for an advanced approach to identify error values that limit the deviation in the reconstructed output, even for the more promising non-NTT operations.

Solution (3). We determine the error value e that fully limits, as in minimizes, the deviation in the reconstructed output through mathematical analysis as follows.

Let $q_1, \dots, q_n$ be a set of primes and define

$$Q := \prod_{i \leq n} q_i, \quad M_i := \frac{M}{q_i}, \quad M_i^{-1} M_i = 1 \bmod q_i.$$

Then, for any integer e ,

$$M_n^{-1} e = 1 \bmod q_n \iff M_n M_n^{-1} e = M_n \bmod Q.$$

This is the smallest non-zero error. Moreover, $M_n \bmod q_n$ is one such integer.

Proof. As indicated, if the fault we introduce is following an integer e , then the resultant error in reconstructed output is $M_n M_n^{-1} e \bmod Q$. Since $M_n^{-1} e = 1 \bmod q_n$, it holds that $M_n^{-1} e = k q_n + 1$, where k is an integer. Thus,

$$M_n M_n^{-1} e = M_n (k q_n + 1) = k M_n q_n + M_n = k M + M_n \quad (9)$$

which implies $M_n M_n^{-1} E = M_n \bmod M$.

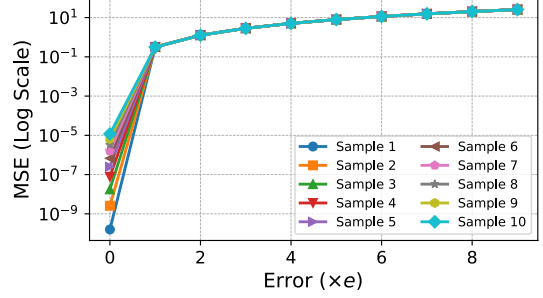


Figure 5: Effect of varying errors on data corruption for CKKS-based polynomial evaluation for 10 sample inputs.

Now, for an integer P with $M_n^{-1} P > 1 \bmod q_n$, it holds that

$$M_n M_n^{-1} P = M_n (k q_n + (M_n^{-1} P \bmod q_n)) \quad (10)$$

$$= k M + M_n (M_n^{-1} P \bmod q_n) > k M + M_n \quad (11)$$

and, thus, $M_n M_n^{-1} P > M_n \bmod M$. Furthermore, if $M_n^{-1} P = 0 \bmod q_n$, then $M_n M_n^{-1} P = M_n k q_n = k Q$ and, thus, $M_n M_n^{-1} P = 0 \bmod Q$.

Finally, we verify that $M_n \bmod q_n$ is one such integer. Indeed, by definition of M_n^{-1} , we have

$$M_n^{-1} (M_n \bmod q_n) \bmod q_n = M_n^{-1} M_n \bmod q_n = 1 \quad (12)$$

Thus, the error value e that results in minimum error propagation toward the output is $M_n \bmod q_n$. $\square$

Note that we showcased the effect of this carefully crafted error in Sec. 3, Fig. 3(c). There, we added the derived error $M_n \bmod q_n$ during the rescaling operation for the last modulus.

4.6 From Injection of Minimal Errors Toward Controlled Data Corruption

Challenge (4). Although the minimum error that can be introduced has been determined by Solution (3), given that the outputs remain in the encrypted ciphertext domain, one challenge remains: the final impact of the corresponding FIAs on the data integrity needs to be validated.

Solution (4). To address this challenge, we perform an empirical analysis using sample plaintext inputs for the application under attack. First, we randomly select plaintext inputs within valid ranges (i.e., confirming to possible values for the application under attack) and encrypt them using the same parameters as the client. Then, we determine the minimal error e as described above and evaluate its end-to-end impact on the computation outputs as follows. While gradually increasing the error scale (by multiplying it with simple integer factors p), we assess the average MSE between the nominal outputs (i.e., without faults induced) and the corrupted outputs for all the sample inputs. Once the error reaches a level that causes sufficient output corruption as desired by the attacker, we finalize the induced error ($p \times e$).

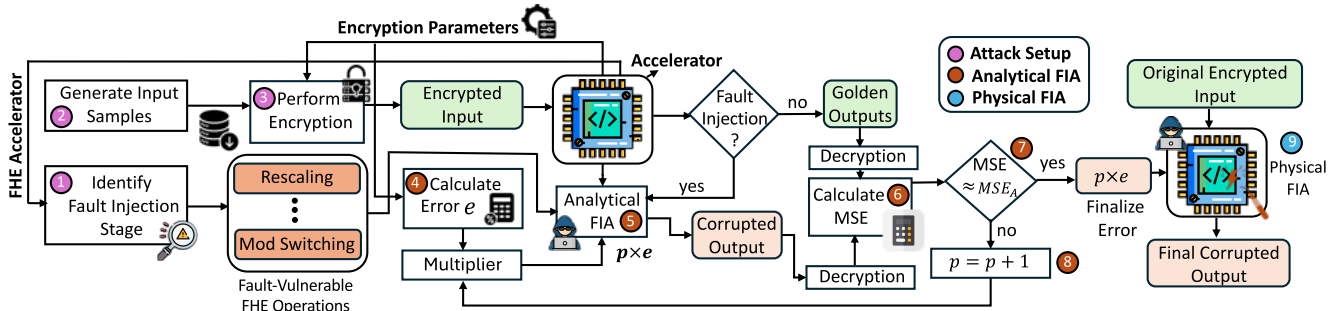


Figure 6: Overall flow for GlitchFHE, followed by actual FIAs in the real-world.

In Fig. 5, we provide a corresponding, exemplary plot for output corruption over increasing error scales. First, note that the different MSE values for zero errors indicate on the nominal error distribution induced by FHE evaluation. Second, outside of the nominal distribution, the output corruption does not vary for the different error scales. This confirms that controlling data corruption is possible across various inputs. Note that, for other applications and parameters, the progression of output corruption scale scale differently. Thus, such an end-to-end validation of actual output corruption is required toward successful attacks in the real world.

4.7 Putting It All Together

Fig. 6 illustrates the end-to-end flow for GlitchFHE, including actual FIAs on FHE accelerators. For a given FHE application, as described above in Solution (2), we first identify the specific stage(s) where fault injection should occur (Step (1)).³ Next, in Steps (2) and (3), we sample plaintext inputs that are selected and encrypted using the same parameters as with the client’s setup, which was described in Solution (4). Next, through Steps (5) to (8), we perform analytical FHE where the error e is induced at the chosen stage, while being carefully increased, and its impact on the final output of the FHE accelerator is assessed. Once the desired level of output corruption (MSE_A) is achieved for the sample inputs, the error value $p \times e$ is finalized, and the analytical part covered by GlitchFHE concludes. Finally, actual FIAs can proceed on the original ciphertext inputs within the FHE accelerator (Step (9)).

5 Experimental Investigation

Through a set of real-world case studies, we demonstrate the effectiveness of GlitchFHE in carefully corrupting encrypted data across various FHE applications, including image processing, neural network (NN)-based image classification, and polynomial evaluation. Our analysis covers the prominent CKKS and BFV schemes. We showcase successful attacks

³In Appendix C, we summarize again the rationale for fault injection in NTT vs coefficient domain.

both at the software level and the physical level. All case studies and findings are a first in the literature.

5.1 Setup

Software Setup. We conduct our experiments on a workstation with an Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz, 16GB of RAM, and running Ubuntu 22.04 OS. Without loss of generality, our experiments utilize the Microsoft SEAL library [43], a widely used C++ open-source library that implements various FHE techniques. We implement GlitchFHE on top of the Microsoft SEAL library. We have performed experiments on the 3.7 and 3.6 versions of SEAL, to demonstrate the portability of our method. Additionally, we utilize Microsoft EVA as FHE compiler.

Applications and Benchmarks. We conduct experiments on representative applications, namely (i) polynomial evaluation and (ii) *CryptoNets* [25], a state-of-the-art, FHE-based accelerator for NN applications. For (i), the benchmarks related to polynomial evaluation for CKKS and BFV are publicly available alongside the Microsoft SEAL library in [43]. For (ii), we utilize the well-known MNIST benchmark for image classification. Further, for CryptoNets on CKKS and BFV, we modify the implementation in [44, 45].

Physical Setup. We obtain a proven ASIC for FHE acceleration, called CoFHEE [26], as courtesy from the authors. Using the Python API provided for CoFHEE [35], we devise the instruction sequence for the last-level rescaling operation in CKKS, which is commonly reducing the ciphertext polynomials from RNS towers of 3 to 2. Considering the limits in CoFHEE, we restrict all the polynomials’ degree to 2^{13} ; this degree is sufficient for any practical FHE application.

For EM FIAs, we utilize a dedicated setup by Langer, namely the ICS 105, a high-precision 4-axis probing station, along with the ICI HH 500-15, an active high-resolution EM probe that enables FIAs focused down to around $200\mu m$ [46]. The probe is driven by the BPS 202, a programmable burst power station. We also use the ICR HH 150-27, a passive high-resolution EM probe that enables side-channel analysis (SCA) of computational activities down to $100\mu m$. We use the Langer provided software for control operations. A

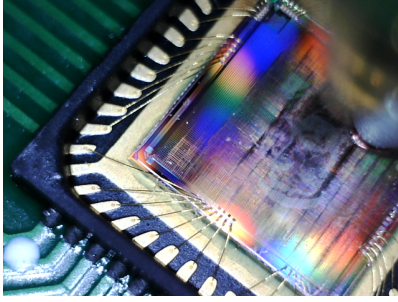


Figure 7: CoFHEE ASIC under the Langer EM FIA probe. Note that the camera’s focus is on the ASIC; thus, only the tip of the probe, which is closest to the ASIC surface, can be discerned to some degree in the upper right corner.

snapshot for the CoFHEE ASIC under the EM FIA probe is shown in Fig. 7, whereas a high-level view on the general setup is shown in Fig. 16 in the appendix. Finally, we also use a Tektronix oscilloscope and probes for signal analysis.

Metrics. We utilize the following metrics to quantify the outcome of FIAs on FHE accelerators.

1. *MSE*: This measures the average squared difference between the corrupted outputs and the actual outputs of the homomorphic computation. Note that MSE is also used by established libraries like SEAL to quantify the difference between homomorphic computed outputs and plaintext outputs [43].
2. *Accuracy*: In ML applications, this measures the percentage of correctly predicted instances.

5.2 Attacking CKKS Applications

As discussed in Sec. 4.4 and following, we perform FIAs during the rescaling operation of CKKS. Here, we perform FIAs at the software level using the proposed *GlitchFHE* framework. In Sec. 5.4, we reconfirm these analytical findings for real-world FIAs attacks using our physical setup.

5.2.1 Polynomial Computation

Setting. We target homomorphic operations evaluating the polynomial $3.14x^3 + 0.4x + 1$. The encryption parameters used are: (i) a polynomial modulus degree of 8192, and (ii) coefficient moduli with sizes of $[60, 40, 40, 60]$. Note that this polynomial and parameters are defined as representative benchmarks along with the SEAL library.

We consider various FIAs scenarios, in particular for the scale of the induced errors. We evaluate the FIAs’ impact on corruption of the final data output using MSE. The golden MSE without any FIAs, i.e., the MSE between homomorphic computed outputs and the plaintext outputs without any fault injection, is 1.31×10^{-12} . We consider data corruption to be stealthy when the change in MSE is ≈ 0 compared to expected outputs, whereas we consider it to be excessive if

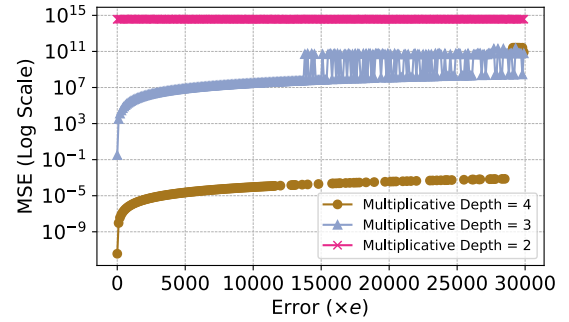


Figure 8: Effect of varying errors on output corruption for CKKS-based polynomial evaluation.

$MSE > 1000$ and subtle if $MSE \leq 1000$. We consider the same setting for other runs on polynomial computation.

Effect of Error Scale. Following the proposed analysis method of *GlitchFHE*, we determine the error e that results in minimal output corruption. We induced this error to the intermediate output in non-NTT domain during the rescaling operation, specifically after the final multiplication operation (depth of four) during polynomial evaluation. This setting led to an MSE of 3.67×10^{-12} , which is very close to the golden MSE—FIAs operating at this minimal error remain stealthy.

Next, to systematically study the relation between error magnitude and output corruption, we varied the error scale from e to $30000 \times e$. The results are shown in Fig. 8; refer to multiplicative depth of four. As expected, the MSE increases with the errors. The shown trend is important for real-world adversaries that seek to carefully tune their attack efforts.

Effect of Multiplicative Depth. This analysis offers insights into the impact of FIAs at different multiplicative depths, i.e., different points in time of the FHE evaluation. Figure 8 shows that output corruption increases once errors are introduced at lower multiplicative depths. This confirms that faults injected early on during the computation propagate through successive operations, potentially over-amplifying their glitching impact on the final output. Accordingly, attackers should apply FIAs in the last level.

5.2.2 Image Classification

Setting. As indicated, we target a state-of-the-art FHE-based NN accelerator called *CryptoNets* [25], which performs the exemplary task of image classification on the MNIST dataset. The default encryption parameters used are: (i) a polynomial modulus degree of 32,768, and (ii) coefficient moduli with sizes $[60, 50, 40, 60, 60, 60, 60, 60]$. The golden/baseline accuracy of the NN is 95.78%. Note that, for FIAs, we consider the attack to remain stealthy when the change in accuracy is 0 when compared to expected outputs, whereas we consider any resulting accuracy $\leq 50\%$ as excessive corruption and any accuracy $> 50\%$ as subtle corruption. We consider the same settings for other experiments on image classification.

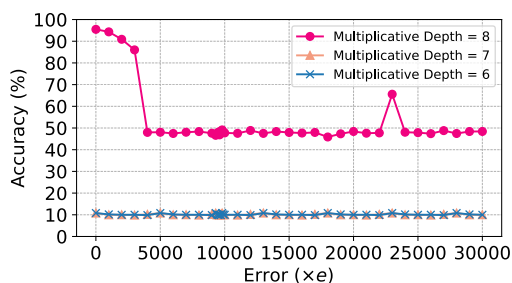


Figure 9: Effect of varying errors on accuracy for CKKS-based CryptoNets implementation for MNIST classification.

Effect of Error Scale. As before, following the proposed analysis method of *GlitchFHE*, we determine the error e that results in minimal output corruption. Then, we systematically varied the error induced in the last rescaling operation (i.e., for multiplicative depth of 8) from e to $30000 \times e$.

Results are shown in Fig. 9. For the minimal error e , the accuracy of CryptoNets is approximately equivalent to the golden accuracy. For FIAs with moderate errors, such as $3000 \times e$, we observe a controlled reduction of accuracy to 86.01%, reaching down around 50% for larger errors. Importantly, the reduction of accuracy is non-linear. We also observe a spike in accuracy for multiplicative depth of 8 and at $\times e = 23000$. Identifying the root-cause is difficult due to the complex interplay of FHE and decryption operations, which can non-linearly amplify or suppress errors. Overall, such analysis enabled by *GlitchFHE* is essential for attackers to find the correct range of errors that corresponds to their desired adversarial impact on the integrity of the application.

Effect of Multiplicative Depth. Errors were introduced at varying depths to examine fault propagation. Shown in Fig. 9, accuracy decreases significantly to around 10% once errors are introduced at any depth below the final level. Conversely, errors introduced at the last level have a limit effect, making output corruption more controllable. Thus, as before, attackers should apply FIAs to the last level.

5.3 Attacking BFV Applications

As discussed in Sec. 4.4 and following, we perform FIAs during modulus switching of BFV. We perform FIAs at the software level using the proposed *GlitchFHE* framework.

5.3.1 Polynomial Computation

Setting. We target BFV-based evaluation of the benchmark polynomial $4(x^2 + 1)(x + 1)^2$. The encryption parameters for this benchmark are: (i) a polynomial modulus degree of 4096, and (ii) coefficient moduli with sizes of [36, 36, 37]. Along with MSE, we also quantify the impact on the noise budget.⁴

⁴For BFV, the noise budget provides some margin before decryption fails; see also Appendix A. While not proposed against FIAs as such, understanding its role for BFV’s resilience (or lack thereof) against FIAs is important.

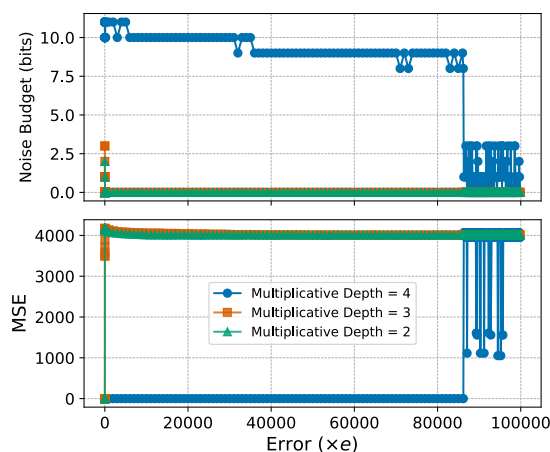


Figure 10: Effect of varying errors on noise budget and on output corruption for BFV-based polynomial evaluation.

Effect of Error Scale. As before, we determine the minimal error and then scale it up for analysing its impact. Results for fault injection into the final layer/depth of four are shown in Fig. 10 and discussed next. First, for smaller error scales, these do *not* corrupt the outputs, but rather strain the noise budget of the BFV scheme. As indicated (and detailed in Appendix A), the noise budget in the BFV scheme allows the computed outputs to remain valid despite some growing error, but only until the noise budget was exhausted. This finding indicates that FIAs in BFV operations are more stealthy, as their impact is “masked” initially, but will ultimately still undermine the integrity (once the noise budget is exhausted). At the same time, however, the resulting data corruption is less controllable, as discussed in more detail next.

Effect of Multiplicative Depth. We observe that the noise budget decreases more significantly when errors are introduced at lower multiplicative depths, as shown in Fig. 10. As before, this is due to the error propagation through successive operations at lower depths. Also shown in Fig. 10, once the noise budget approaches zero, the outputs become increasingly erroneous, as expected. This reaffirms the need for careful management of FIAs, through *GlitchFHE*, particularly at lower depths, to prevent the early depletion of the noise budget and limit the output corruption as required.

5.3.2 Image Classification

Setting. As before, we target the MNIST image classification application in CryptoNets. The default encryption parameters for BFV used here are: (i) a polynomial modulus degree of 8,192 and (ii) coefficient moduli with sizes [43, 43, 44, 44, 44]. The golden/baseline accuracy of the NN is 98.07%.

Effect of Error Scale. Results for the final depth are shown in Fig. 11. Note that, even for small-scale errors, the accuracy quickly and significantly dropped to around 50%. This reduction remained stable for medium-scale errors, up until the

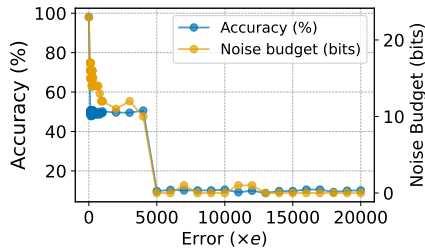


Figure 11: Effect of varying errors on noise budget, accuracy of BFV-based CryptoNets for MNIST image classification.

noise budget is exhausted. Thus, the BFV-based CryptoNets implementation is only somewhat resilient to small errors and not resilient to larger errors. Interestingly, this is despite the concept of noise budget in BFV, which affirms our hypothesis that established error management techniques in FHE schemes are not helpful against FIAs per se.

Effect of Multiplicative Depth. Here, we observe very similar results as before, specifically as in Sec. 5.2.2. For brevity, we delegate the plot and discussion to Sec. E.

5.4 Physical Attack on CKKS with CoFHEE

Setting. Here, we conduct a case study for real-world FIA attacks. While we focus here on the CoFHEE ASIC accelerating the polynomial evaluation in CKKS as in Sec. 5.2.1, the outlined practical challenges and solutions would also apply similarly to other FHE applications and accelerators.

Challenges 1 and 2: Identifying the Right Time and Place for Fault Injection. As shown in this work, the effects of fault injection vary significantly across different FHE operations. Once attackers understand which operation(s) to attack—based on formal reasoning and analytical exploration through *GlitchFHE*—they still need to identify (1) the precise point in time for when and (2) the precise location for where these operations are running on the hardware.

Solution: Power Side-Channel Analysis. Given their complex computation and data management, FHE operations may well be identified by power SCA.⁵ For CKKS rescaling, the operation of interest here, recall its procedure listed in Sec. 2.2: there is one iNTT operation at the beginning and two NTT operations afterwards. Fig. 12 illustrates a corresponding power trace. Note how the iNTT operation exhibits a distinctive and identifiable waveform signature; see Appendix E for further details. This way, Challenge 1 is resolved.

To identify the precise location and, thus, resolve Challenge 2, we utilize high-resolution EM probing, which enables proxy inference for localized power consumption. We were able to pinpoint multiple regions of interest on the CoFHEE ASIC. Presumably, these regions cover the memories and logic utilized for CKKS rescaling. Note that, even without



Figure 12: System-level power profile for CoFHEE during CKKS rescaling operation on three RNS towers. Fault injection is triggered after the iNTT operation.

advanced reverse engineering, memories can be differentiated from logic, namely from the different routing structures on top of those components, as also visible in Fig. 7.

Challenge 3: Controlling the Data Corruption. Depending on the physical mechanisms utilized for FIAs, the scale of induced errors and, thus, the resulting data corruption may greatly vary. Accordingly, for real-world attacks, practical means for error control are required.

Solution: Tuning and Validating the Fault Injection. For EM as fault mechanism, it is common that multiple faults are injected at once, while the location and number of faults induced can be tuned through the EM field. The latter can be controlled via positioning of the probe and through the field-inducing parameters (pulse frequency, voltage, etc.) Naturally, this requires efforts for trial and *error* (pun intended). In our experiments, we find the following.

First, the different locations identified by Solution 2 are not equally sensitive to FIAs. We observe that the memory regions allow for more controllable fault injection than the logic regions. More specifically, we can control fault injection down to the byte-level of coefficients in various memories; this level of accuracy is sufficient for successful FIAs.

Second, the byte-level corruption in the memory depends on the EM probe settings, i.e., probe voltage and distance, as shown in Fig. 13. Here, for each data point, (i) the memory is loaded with a known dataset related to the FHE application under attack, (ii) a fault is injected, (iii) the memory is read back to check for the number of corrupted bytes (if any). This process is repeated ten times, and the majority outcome is plotted. Note that we use single-pulse fault injection here,⁶ as the maximum frequency for fault injection supported by the Langer setup is 20 kHz, which is notably slower than the chip’s operational frequency of 40 MHz. Finally, this FIA approach is generic and applicable to any FHE application under attack, all while the repetitive process is only required for this study on impact of fault-injection settings.

⁵While not explicitly formulated for the threat model (Sec. 4.1), such efforts are readily compatible with the attacker’s resources and capabilities.

⁶Further insights from multiple-fault injection are given in Appendix E.

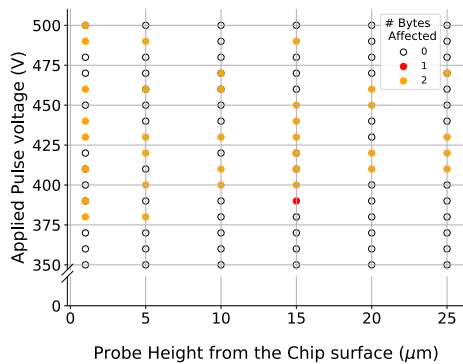


Figure 13: Data corruption by EM fault injection under varying settings for voltage and probe height.

Third, despite the above noted controllable impact, inducing the exact errors identified by `GlitchFHE` can still be difficult. Thus, we require validation of the resulting data corruption. We follow a similar approach as in Sec. 4.6: we monitor the MSE for sample inputs while tuning the FIAs.

In short, here we have demonstrated that FIAs devised by `GlitchFHE` can be successfully applied in the real world.

6 Related Work and Discussion

6.1 Other Attacks on FHE

Chaturvedi et al. [47] analyzed the security of TFHE and FHEW, assuming a malicious cloud server induces decryption errors in ciphertexts and observes client reactions. The attack proposes an *indistinguishably-against-ciphertext-verification attack* model, using timing information and a binary search approach to recover the secret key in full.

Cheon et al. [48] proposed a hybrid attack on the LWE problem, targeting schemes with sparse and ternary secret keys, as found in prominent FHE libraries like HEAAN, HELib, and SEAL. The attack combines dual lattice reduction, which balances dimension and error size, with a man-in-the-middle strategy. By exploiting key sparsity, the attack reduces the computational complexity for large modulus parameters, exposing inherent vulnerabilities in certain FHE schemes.

Chillotti et al. [49] explored SHE and FHE systems, focusing on so-called safe-error and reaction attacks. A malicious cloud provider introduces faults and observes client reactions to extract secret key bits with minimal queries. Vulnerabilities in trans-ciphering and bootstrapping procedures, critical to FHE schemes, are identified as potential attack vectors.

Zhang et al. [50] proposed a reaction attack on FHE schemes, considering black-box, gray-box, and white-box threat models. The attack targets FHE schemes built on Gentry’s framework, such as BGV, DGHV, and GSW. By modifying intermediate ciphertexts and observing client reactions, the attacker can recover encrypted messages and break the secret key with logarithmic time complexity.

Aydin et al. [51] introduced an SCA-driven attack on homomorphic encryption, specifically targeting Microsoft SEAL’s implementation of the BFV protocol. The attack exploits vulnerabilities in the Gaussian sampling sub-routine during the encryption phase, allowing the extraction of the entire message with a single power measurement. Using these measurements, they apply the BKZ algorithm to estimate the search space, reducing the encryption security from 2^{128} to $2^{4.41}$.

6.2 Applicability of `GlitchFHE` for FPGA, GPU, and CPU Accelerators

Recall that `GlitchFHE` consists of two steps: (i) SCA to determine the timing for FIA, and (ii) corruption of the encrypted data residing in memory. The effectiveness of `GlitchFHE` on any FHE accelerator depends on the feasibility of both steps for the underlying hardware platform.

For FPGAs, this process may be straightforward, due to the well-structured and partially isolated computation and memory components in FPGAs. For example, power traces may be more discernible on FPGAs, as the logic for different FHE operations is likely implemented in separate areas with limited interference from other activities. Further, FPGAs operate with fixed clocking, deterministic data-flow scheduling, fixed-function data paths, and domain-specific pipelines [52–55], which should all ease both trace alignment and fault timing. Thus, the insights gained from our ASIC-based case studies may be transferable to FPGA-based implementations of FHE accelerators; this will be scope for future work.

CPUs are expected to introduce some challenges for `GlitchFHE`, due to tightly integrated execution units, shared caches, speculative behavior, etc., which can all obscure precise timing. Although prior work shows that SCA and FIAs are possible on CPUs [37, 56–59], aligning both steps of `GlitchFHE` may be complex. GPUs likely introduce even further challenges, as power traces may be impacted by high-energy off-chip memories, PCIe interfaces, etc., while their massively-parallel execution model also introduces non-determinism through dynamic warp-scheduling and asynchronous execution [60]. Thus, applying `GlitchFHE` to CPUs and GPUs may require the design of timing-aligned FHE workloads, isolating the relevant compute kernels, and carefully tuning the trace acquisition, all to account for platform-specific noises and runtime behavior.

6.3 Potential Countermeasures

Recall that `GlitchFHE` undermines FHE data integrity by means of FIAs. Thus, protecting against `GlitchFHE` requires robust FIA countermeasures. Researchers have proposed several strategies, including hardware-based techniques such as sensor-based detection [61–63], EM shielding [64], and layout hardening [65], as well as software-level defenses like consistency checks [66]. While these approaches may provide

some protection against `GlitchFHE`, they all require careful tailoring for FHE accelerators, and they all incur some cost/overheads. More details are discussed in Appendix D.

6.4 Verifiable FHE

Here, we discuss recent efforts for verifiable FHE (vFHE) and how their constraints prevent them from effectively detecting the faults introduced by `GlitchFHE`. Note that vFHE schemes can be broadly categorized into zero-knowledge proof (ZKP), message authentication code (MAC), and trusted execution environment (TEE)-based techniques. Each class has distinct principles and limitations.

ZKP-based schemes [67, 68] construct proofs by arithmetizing the FHE evaluation circuit, which is computationally intensive and becomes intractable for circuits with high multiplicative depth, as in CKKS or BFV. Moreover, these schemes typically lack support for operations such as rotation or automorphism—crucial for leveraging the single instruction multiple data (SIMD) capabilities of modern FHE schemes. Consequently, their scope is confined to shallow FHE schemes like TFHE [27].

MAC-based schemes, owing to their symmetric cryptographic design, are inherently constrained to single-client settings. While they can offer strong guarantees of correctness and confidentiality in such contexts, they are fundamentally unsuited for multi-party scenarios involving private inputs from different entities. More recently, hybrid methods such as DataSeal [69] have been introduced, combining algorithm-based fault tolerance (ABFT) with FHE to reduce runtime overheads. However, their applicability is limited to specific computations such as matrix multiplication.

Finally, **TEEs** provide hardware-enforced isolation for secure computation. However, TEEs do not offer inherent verifiability, as clients lack the ability to independently validate the correctness of the computation. Furthermore, TEEs are not universally supported across platforms and remain vulnerable to a range of attacks, including fault injection [70].

In summary, while vFHE schemes offer integrity checks, their deployment remains limited due to constraints for real-world FHE applications (Table 3). Thus, in current practice, the integrity of FHE computations is often upheld through trust assumptions, either explicitly by assuming a semi-honest server, or implicitly, by relying on hardware trust models such as TEEs. However, both approaches remain vulnerable to FIAs. Importantly, none of the existing vFHE schemes encompass the class of operations targeted at by `GlitchFHE`.

7 Conclusion

For the first time, we show that FIAs on FHE can manipulate encrypted data during outsourced computation, effectively undermining the integrity of FHE applications. We present

Table 3: Comparison of vFHE schemes based on real-world deployment requirement for FHE. Note that ‘•’ or ‘-’ indicates that the feature exists or does not exist, respectively.

Scheme	Complex FHE Operations	Multi-Client Support	Platform-Independent
FHE + ZKP	-	•	•
FHE + MAC	•	-	•
FHE + TEE	•	•	-

`GlitchFHE`, an analytical framework that enables security researchers to glitch encrypted data in different formats and cause controllable corruption in the final outputs. We demonstrate the effectiveness of FIAs guided by `GlitchFHE` through a range of case studies, covering different FHE schemes and applications. We also confirm the feasibility of such attacks using an EM FIA setup on CoFHEE, a state-of-the-art ASIC FHE accelerator. For both CKKS and BFV-based implementations of standard applications like polynomial evaluation and NNs for image classification, we show how data corruption evolves for varying errors. For example, already for few multiples of the analytically derived minimal errors, we observe a notable (around 10%) but controllable reduction in accuracy.

We envision that our work and the related release provides a valuable resource for FHE R&D. Again, we highlight the importance of considering data integrity threats for FHE and developing effective countermeasures. Thus, for future work, the extension of error-handling measures, including verifiable schemes and others, toward full protection against various FIAs demonstrated in this work, is highly warranted.

Ethics Considerations

To the best of our understanding, the *USENIX Security’25 Ethics Guidelines* do not represent any major concern for our work. Further considerations are discussed next.

Potential for Misuse. Our findings could be misused by malicious actors to conduct attacks against FHE systems. This could lead to data corruption, undermining the trust and reliability of FHE applications. Through this paper, we are transparent about the related findings.

Impact on FHE Development. By highlighting potential weaknesses in FHE, our work contributes to ongoing research. We believe that openly discussing vulnerabilities is essential for building more robust and secure FHE systems.

Responsible Disclosure. We are committed to responsible disclosure practices and will engage with relevant stakeholders to ensure that our findings are communicated effectively and ethically. This includes coordinating with FHE developers and security researchers to facilitate the development of mitigations before widespread publication. More specifically, since our attack approach is generic and not limited to specific FHE implementations and vendor, we aim to address this through community outreach in general.

Open Science

We make artifacts for both our software and hardware setup available at <https://doi.org/10.5281/zenodo.15615934>. These materials include source codes, configuration files, documentation, and instructions and requirements for installation. More specifically, we release the CKKS and BFV implementations of (i) polynomial evaluation (Sec. 5.2.1, Sec. 5.3.1), and (ii) image classification (Sec. 5.2.2, Sec. 5.3.2), all tailored as needed for GlitchFHE. For the physical experiments (Sec. 5.4), we release (i) setup details of the representative FHE accelerators under attack i.e., the CoFHEE chip [26], (ii) Python APIs for executing various polynomial operations for FHE evaluation on the CoFHEE chip, and (iii) the documentation of the EM-based FIA setup.

Acknowledgments

This work was supported by the Center for Cyber Security Abu Dhabi (CCSAD) and in part by the Defense Advanced Research Projects Agency (DARPA) under the DPRIVE program, contract no. HR0011-21-9-0003.

References

- [1] D. Sears. Cloud Hosting and Cloud Computing Statistics for 2024 . <https://www.bluehost.com/blog/cloud-hosting-statistics/>, 2024.
- [2] D. Archer et al. Applications of homomorphic encryption. In *Crypto Standardization Workshop, Microsoft Research*, volume 14, pp. 1–14, 2017.
- [3] Ö. Kocabaş and T. Soyata. Medical data analytics in the cloud using homomorphic encryption. In *Handbook of Research on Cloud Infrastructures for Big Data Analytics*, pp. 471–488. 2014.
- [4] O. Kocabas et al. Assessment of cloud-based health monitoring using homomorphic encryption. In *Proc. Int. Conf. Comp. Des.*, pp. 443–446, 2013.
- [5] IBM. IBM Cost of a Data Breach Report 2023. <https://www.ibm.com/reports/data-breach>, 2023.
- [6] C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.
- [7] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption, 2012. <https://eprint.iacr.org/2012/144>.
- [8] Z. Brakerski et al. Fully homomorphic encryption without bootstrapping, 2011. <https://eprint.iacr.org/2011/277>.
- [9] J. H. Cheon et al. Homomorphic encryption for arithmetic of approximate numbers. In *Proc. ASIACRYPT*, pp. 409–437, 2017.
- [10] M. Blatt et al. Secure large-scale genome-wide association studies using homomorphic encryption. *Proc. Nation. Acad. Sci.*, 117(21):11608–11613, 2020.
- [11] E. Sarkar et al. Fast and scalable private genotype imputation using machine learning and partially homomorphic encryption. *IEEE Access*, 9:93097–93110, 2021.
- [12] J. Bos et al. Private predictive analysis on encrypted medical data. *J. Biomed. Inf.*, 50, 05 2014.
- [13] E. Sarkar et al. Privacy-preserving cancer type prediction with homomorphic encryption. *Sci. Rep.*, 13(1), 2023.
- [14] F. G. Mármol et al. Do not snoop my habits: preserving privacy in the smart grid. *IEEE Communications Magazine*, 50(5):166–172, 2012.
- [15] Programming Computation on Encrypted Data . <https://www.darpa.mil/program/programming-computation-on-encrypted-data>.
- [16] DARPA Selects Researchers to Accelerate Use of Fully Homomorphic Encryption. <https://www.darpa.mil/news-events/2021-03-08>, 2021.
- [17] ElectionGuard. <https://www.electionguard.vote/>.
- [18] L. Kristin et al. Password Monitor: Safeguarding passwords in Microsoft Edge. <https://www.microsoft.com/en-us/research/blog/password-monitor-safeguarding-passwords-in-microsoft-edge/>, 2021.
- [19] N. Samardzic et al. Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data. In *Proc. Int. Symp. Comp. Archit.*, pp. 173–187, 2022.
- [20] M. Yan. Accelerating Fully Homomorphic Encryption with an Open Source FPGA Library. <https://www.intel.com/content/www/us/en/developer/articles/technical/homomorphic-encryption/accelerating-homomorphic-encryption-for-fpga.html>, 2022.
- [21] Z. Peng. Danger of using fully homomorphic encryption: A look at microsoft seal. <https://arxiv.org/abs/1906.07127>, 2019.
- [22] J. H. Cheon et al. Attacks against the IND-CPAD security of exact FHE schemes. In *Proc. Comp. Comm. Sec.*, pp. 2505–2519, 2024.

- [23] B. Wang et al. Cryptanalysis of a symmetric fully homomorphic encryption scheme. *Trans. Inf. Forens. Sec.*, 13(6):1460–1467, 2018.
- [24] G. Chunsheng. Attack on fully homomorphic encryption over the integers. <https://arxiv.org/abs/1202.3321>, 2012.
- [25] R. Gilad-Bachrach et al. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proc. Int. Conf. ML*, pp. 201–210, 2016.
- [26] M. Nabeel et al. Cofhee: A co-processor for fully homomorphic encryption execution. In *Proc. Des. Autom. Test Europe*, pp. 1–2, 2023.
- [27] I. Chillotti et al. Tfhe: fast fully homomorphic encryption over the torus. *J. Cryptology*, 33(1):34–91, 2020.
- [28] A. Al Badawi and Y. Polyakov. Demystifying bootstrapping in fully homomorphic encryption. <https://eprint.iacr.org/2023/149>, 2023.
- [29] M. Albrecht et al. Homomorphic encryption standard. In *Protecting privacy through homomorphic encryption*, pp. 31–62. Springer, 2021.
- [30] E. Flapan et al. *Number Theory: A Lively Introduction with Proofs, Applications, and Stories*. 2010.
- [31] J. H. Cheon et al. A full rns variant of approximate homomorphic encryption. In *Proc. Sel. Areas Cryptogr.*, pp. 347–368, 2019.
- [32] F. Boemer et al. Intel HEXL: accelerating homomorphic encryption with intel avx512-ifma52. In *Proc. Encr. Comput. & Appl. Hom. Cryptogr.*, pp. 57–62, 2021.
- [33] W. Jung et al. Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with gpus. *Trans. Cryptogr. Hardw. Embed. Sys.*, pp. 114–148, 2021.
- [34] J. Zhang et al. Sok: Fully homomorphic encryption accelerators. *Comp. Surv.*, 56(12):1–32, 2024.
- [35] MomaLab. Cofhee: Hardware accelerator for fhe. <https://github.com/momalab/CoFHEE>, 2025.
- [36] B. Selmke et al. Attack on a DFA protected AES by simultaneous laser fault injections. In *Proc. Worksh. Fault Diag. Tol. Cryptogr.*, pp. 36–46, 2016.
- [37] A. Cui and R. Housley. BADFET: Defeating modern secure boot using second-order pulsed electromagnetic fault injection. In *Proc. WS Off. Tech.*, 2017.
- [38] P. Bayon et al. Fault model of electromagnetic attacks targeting ring oscillator-based true random number generators. *J. Cryptogr. Eng.*, 6(1):61–74, 2016.
- [39] A. Dehbaoui et al. Electromagnetic transient faults injection on a hardware and a software implementations of AES. In *Proc. Worksh. Fault Diag. Tol. Cryptogr.*, pp. 7–15, 2012.
- [40] V. van der Veen et al. Drammer: Deterministic rowhammer attacks on mobile platforms. In *Proc. Comp. Comm. Sec.*, pp. 1675–1689, 2016.
- [41] P. Qiu et al. VoltJockey: Breaching TrustZone by software-controlled voltage manipulation over multi-core frequencies. In *Proc. Comp. Comm. Sec.*, pp. 195–209, 2019.
- [42] M. Eslami et al. A survey on fault injection methods of digital integrated circuits. *Integr.*, 71:154–163, 2020.
- [43] Microsoft SEAL. <https://github.com/Microsoft/SEAL>, November 2020.
- [44] E. Chielle et al. Optimizing ciphertext management for faster fully homomorphic encryption computation. In *Proc. Des. Autom. Test Europe*, pp. 1–6, 2024.
- [45] E. Chielle et al. Furbo. <https://github.com/momalab/furbo>, 2024.
- [46] Langer EMV-Technik GmbH. Ici hh500-15 l eft set – pulse magnetic field source set. <https://www.langer-emv.de/en/product/fault-injection/116/ici-hh500-15-l-eft-set-pulse-magnetic-field-source-set/1165>.
- [47] B. Chaturvedi et al. A practical full key recovery attack on TFHE and FHEW by inducing decryption errors. <https://eprint.iacr.org/2022/1563>, 2022.
- [48] J. H. Cheon et al. A hybrid of dual and meet-in-the-middle attack on sparse and ternary secret lwe. *IEEE Access*, 7:89497–89506, 2019.
- [49] I. Chillotti et al. Attacking FHE-based applications by software fault injections. <https://eprint.iacr.org/2016/1164>, 2016.
- [50] Z. Zhang et al. Reaction attack on outsourced computing with fully homomorphic encryption schemes. In *Proc. Int. Conf. Inform. Comm. Sec.*, pp. 419–436, 2012.
- [51] F. Aydin et al. Reveal: Single-trace side-channel leakage of the seal homomorphic encryption library. In *Proc. Des. Autom. Test Europe*, pp. 1527–1532, 2022.
- [52] M. S. Riazi et al. Heax: An architecture for computing on encrypted data. In *Proc. Arch. Supp. Programm. Lang. Op. Sys.*, pp. 1295–1309, 2020.
- [53] F. Turan et al. Heaws: An accelerator for homomorphic encryption on the amazon aws fpga. *Trans. Comp.*, 69(8):1185–1196, 2020.

- [54] A. C. Mert et al. Design and implementation of encryption/decryption architectures for bfv homomorphic encryption scheme. *Trans. VLSI Syst.*, 28(2):353–362, 2020.
- [55] D. B. Cousins et al. Designing an fpga-accelerated homomorphic encryption co-processor. *Trans. Emerg. Top. Comp.*, 5(2):193–206, 2017.
- [56] F. Aydin and A. Aysu. Leaking secrets in homomorphic encryption with side-channel attacks. <https://eprint.iacr.org/2023/1128>, 2023.
- [57] Z. Chen et al. VoltPillager: Hardware-based fault injection attacks against intel SGX enclaves using the SVID voltage scaling interface. In *Proc. USENIX Sec. Symp.*, pp. 699–716, 2021.
- [58] J. Delvaux et al. Breaking Espressif’s ESP32 v3: Program counter control with computed values using fault injection. In *Proc. WS Off. Tech.*, pp. 229–243, 2024.
- [59] Y. Wang et al. Hertzbleed: Turning power Side-Channel attacks into remote timing attacks on x86. In *Proc. USENIX Sec. Symp.*, pp. 679–697, 2022.
- [60] C. Luo et al. Side-channel power analysis of a gpu aes implementation. pp. 281–288, 2015.
- [61] N. Homma et al. Em attack is non-invasive? design methodology and validity verification of em attack sensor. In *Trans. Cryptogr. Hardw. Embed. Sys.*, pp. 1–16, 2014.
- [62] P. Ravi et al. Ppap and ippap: PII-based protection against physical attacks. In *Proc. Comp. Soc. Symp. VLSI*, pp. 620–625, 2018.
- [63] M. R. Muttaki et al. Ftc: A universal framework for fault-injection attack detection and prevention. *Trans. VLSI Syst.*, 32(7):1311–1324, 2024.
- [64] H. Wang et al. A physical design flow against front-side probing attacks by internal shielding. *Trans. Comp.-Aided Des. Integr. Circ. Sys.*, 39(10):2152–2165, 2020.
- [65] J. Bhandari et al. Defending integrated circuit layouts. *Trans. Inf. Forens. Sec.*, 20:46–59, 2025.
- [66] T. Schneider et al. Parti – towards combined hardware countermeasures against side-channel and fault-injection attacks. In *Proc. CRYPTO*, pp. 302–332, 2016.
- [67] C. Ganesh et al. Rinocchio: Snarks for ring arithmetic. *J. Cryptology*, 36(4):41, 2023.
- [68] C. Knabenhans et al. vfhe: Verifiable fully homomorphic encryption. In *Proc. Encr. Comput. & Appl. Hom. Cryptogr.*, pp. 11–22, 2023.
- [69] M. H. Santriaji et al. Datasael: Ensuring the verifiability of private computation on encrypted data. *arXiv:2410.15215*.
- [70] S. Van Schaik et al. Sok: Sgx.fail: How stuff gets exposed. In *Proc. Symp. Sec. Priv.*, pp. 4143–4162, 2024.
- [71] A. Kim et al. Revisiting homomorphic encryption schemes for finite fields. In *Proc. ASIACRYPT*, pp. 608–639, 2021.
- [72] A. Kurian et al. Tpxtract: An exhaustive hyperparameter extraction framework. *Trans. Cryptogr. Hardw. Embed. Sys.*, 2025(1):78–103.
- [73] Keysight Technologies. DS1002A Pattern Based Trigger Generator. <https://www.keysight.com/us/en/product/DS1002A/pattern-based-trigger-generator.html>, 2025.
- [74] Langer EMV-Technik GmbH. Ics 105 set – ic scanner 4-axis positioning system. <https://www.langer-emv.com/en/product/langer-scanner/41/ics-105-set-ic-scanner-4-axis-positioning-system/144>, 2025.
- [75] NewAE Technology Inc. Chipwhisperer-husky. <https://www.newae.com/product-page/chipwhisperer-husky>, 2025.

A BFV and BGV

Overview. Like CKKS, both the BFV [7] and BGV [8] schemes are built on a RLWE problem. Unlike CKKS, however, recall that their message space is integers. More specifically, the plaintext and ciphertext spaces in BFV and BGV are defined over two distinct polynomial rings denoted by $P = R_t = \mathbb{Z}_t[X]/(X^n + 1)$ and $C = R_Q \times R_Q$, where $R_Q = \mathbb{Z}_Q[X]/(X^n + 1)$, $n \in \mathbb{Z}$ is the ring dimension, and $t \in \mathbb{Z}$ and $Q \in \mathbb{Z}$ are the plaintext and ciphertext coefficients, respectively, and $Q \gg t$. The main difference between BFV and BGV is in how the message is encoded, as elaborated next.

Message Encoding and Decoding. Similar to CKKS, a popular scheme for BFV and BGV is evaluation-based encoding, where the message vector $\mathbf{m}$ is seen as the evaluation points of a polynomial $p(x)$. Evaluation points are the roots of the unity of $x^n + 1$. In other words, encoding is nothing but iNTT of the message vector, i.e., $p(x) = iNTT(\mathbf{m})$.

BGV encodes the message in the least significant bits of the plaintext polynomial coefficients, while BFV stores in the most significant bits. Thus, for BFV, the coefficients of $p(x)$ are scaled with factor $\Delta = \lfloor Q/t \rfloor$, to provide some budget for noise addition during encryption and FHE evaluation. The higher the value of Q , the larger the budget for the noise to grow, implying higher multiplicative depth for FHE evaluation. For BGV, there is no such scaling, but during encryption,

for the noise addition, the error polynomial $\mathbf{e}$ is scaled by the plaintext modulus t and then added, i.e., $t\mathbf{e}$ is added as noise. This differs from BFV and CKKS, where the sampled error polynomial is added directly without any scaling.

Ciphertext Format. Like CKKS, BFV and BGV also follow the RLWE where the ciphertext $\mathbf{c}$ is a pair of polynomials $(\mathbf{a}, \mathbf{b}) \in R_Q = \mathbb{Z}_Q[X]/(X^N + 1)$. For BFV, decryption of $\mathbf{c}$ with the secret key $\mathbf{s}$ follows $\text{Dec}(\mathbf{c}) = \mathbf{b} + \mathbf{a} \cdot \mathbf{s} = [\Delta \mathbf{m} + \mathbf{e}]_Q$ where $\mathbf{m} = \llbracket [\Delta \mathbf{m} + \mathbf{e}]_Q / \Delta \rrbracket_t$. For BGV, decryption follows $\text{Dec}(\mathbf{c}) = \mathbf{b} + \mathbf{a} \cdot \mathbf{s} = [\mathbf{m} + t\mathbf{e}]_Q$ where $\mathbf{m} = \llbracket [\mathbf{m} + t\mathbf{e}]_Q \rrbracket_t$. Note that the value of ciphertext modulus Q has to be very large to support higher multiplicative depth. So, as with CKKS, RNS is used here to split the ciphertext polynomials into multiple polynomials with coefficient modulus q_i such that $Q = \prod q_i$.

For both BGV and BFV, for the decryption to work correctly, one needs to ensure that $\frac{t}{Q} \|\mathbf{e}\|_\infty < 1/2$, else the noise will overflow and corrupt the message; more related details are discussed in the remainder.

For BGV, observe from the decryption formulation that, if two ciphertexts are multiplied, noise growth is quadric to the plaintext modulus t . This noise is controlled by modulus switching similar to rescaling. Hence, like CKKS, BGV also has a set of moduli $Q_0, Q_1, \dots, Q_L$ where Q_i divides Q_{i+1} .

A ciphertext starts with modulus Q_L and, after each multiplication, the ciphertext's coefficients are rescaled by factor Q_i/Q_{i+1} and the modulus is switched from Q_{i+1} to Q_i . Thus, the higher the Q_L , the more multiplications can be supported without bootstrapping. BFV also uses the modulus-switching operation during multiplication between two ciphertexts [71].

From the decryption formulation for BFV provided above, it can be seen that multiplication between two ciphertexts $\mathbf{c}$ and $\mathbf{c}'$ results in scaling of corresponding message $[m \cdot m']_t$ by Δ^2 . The main consequence of this scaling is the product of $\mathbf{c}$ and $\mathbf{c}'$ cannot be reduced modulo Q . Therefore, the multiplication is done in R_{QP} , with $P > Q$, followed by a scale-down operation with $t/Q \approx \Delta^{-1}$ to remove the extra Δ factor, then again switching back to R_Q . In other words, before multiplication between two ciphertexts, the corresponding ciphertexts' polynomials have to undergo modulus switching twice, namely first to switch the modulus from Q to QP and, after multiplication and scale-down operation, to switch the modulus back to Q from PQ .

B Zero-Sum Property of iNTT Matrix

Recall the generic iNTT matrix from Equation 4, or rather consider an exemplary 4×4 matrix $\mathbf{M}_{\text{iNTT}}$ as follows:

$$\mathbf{M}_{\text{iNTT}} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \omega^{-3} \\ 1 & \omega^{-2} & \omega^{-4} & \omega^{-6} \\ 1 & \omega^{-3} & \omega^{-6} & \omega^{-9} \end{bmatrix} \quad (13)$$

Here, ω is a primitive 4th root of unity modulo a prime p , satisfying $\omega^4 \equiv 1 \pmod{p}$. Using this, powers of ω can be

reduced modulo 4 in the exponent, and the following generalized properties hold:

$$\begin{aligned} \omega^{k+N} &= \omega^k, \quad \forall k \in \mathbb{Z}, \quad \Rightarrow \quad \omega^{k+4} = \omega^k \quad \text{for } N = 4, \\ \omega^{\frac{N}{2}} &= -1 \quad \text{if } N \text{ is even}, \quad \Rightarrow \quad \omega^2 = -1 \quad \text{for } N = 4, \\ \omega^{N-(k \bmod N)} &= \omega^{-k} \quad \Rightarrow \quad \omega^3 = \omega^{-1} \quad \text{for } N = 4. \end{aligned}$$

In turn, this transforms $\mathbf{M}_{\text{iNTT}}$ to:

$$\mathbf{M}_{\text{iNTT}} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -\omega & -1 & \omega \\ 1 & -1 & 1 & -1 \\ 1 & \omega & -1 & -\omega \end{bmatrix} \quad (14)$$

Now, we can observe that the sum of each row except the first one is 0, proving the zero-sum property.

C Rationale and Trade-offs for Fault Injection in NTT vs Coefficient Domain

Recall that FHE computations are performed in the NTT domain, because of the complexity for polynomial multiplication reducing there from $O(N^2)$ to $O(N \log N)$. However, the iNTT transformation causes wide diffusion of any fault injected in the NTT domain (Equation 2). Thus, to achieve some limited, localized errors, fault injection requires modifying all elements of a ciphertext polynomial simultaneously with identical errors (Sec. 4.3). This makes FIAs in the NTT domain prohibitively challenging in practice.

In contrast, fault injection in the coefficient domain does not require such uniformity and is more controllable, but is only applicable during a limited subset of operations that perform intermediate iNTT transformations, such as rescaling and modulus switching (Table 2 and Sec. 4.4). Furthermore, considering that (1) ciphertexts reside in the RNS domain and (2) typically only the last modulus component is transformed back to the coefficient domain, the scope for FIAs is constrained to this final modulus. Importantly, errors introduced at this stage may induce larger impacts than desired due to the CRT-based reconstruction (Sec. 4.5). We address this challenge by mathematically deriving a minimal-impact fault value, $M_n \bmod q_n$, that leads to bounded corruption in the reconstructed plaintext (Sec. 4.5).

D Potential Countermeasures

Sensor-based Detection employs circuitry such as ring oscillators [61], phase detectors [62], and fault-to-time converters [63] to detect faults by monitoring voltage fluctuations or timing anomalies in the chip. However, integrating such sensors often (i) incurs significant performance overheads, and (ii) could be identified by attackers and potentially disabled through targeted FIA. Consequently, there remains a critical need for more advanced sensor-based countermeasures.

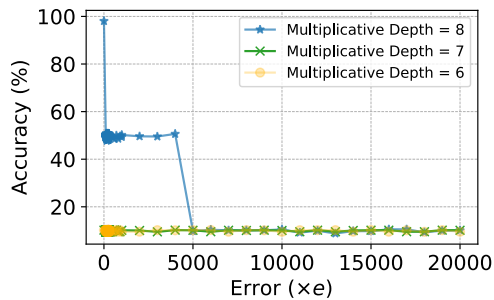


Figure 14: Effect of varying errors on accuracy for BFV-based CryptoNets implementation for MNIST image classification, across different multiplicative depths.

EM Shielding mitigates unwanted EM interference using conductive barriers, commonly realized through metal layers in the chip’s packaging or routing layers [64]. These barriers reflect/absorb incident EM waves to protect sensitive circuitry. Shielding performance depends on EM radiation frequency and material properties, necessitating customized strategies for specific FHE accelerators.

Layout Hardening involves strategic placement and routing of critical logic and data paths to reduce susceptibility to localized EM-based FIA [65]. As with other countermeasures, there are trade-offs between shielding and overheads.

Consistency Checks improve reliability of critical data, such as intermediate outputs, by adding redundancy via encoding schemes like Hamming codes. In *GlitchFHE*, error correction codes (ECC) could help maintain data integrity by detecting and correcting EM-induced bit-flips. Though effective against random faults, ECCs incur area/performance overhead and may be bypassed by precise FIAs [66].

In Summary, all the aforementioned countermeasures (i) target hardware like cryptographic and ML accelerators, and not FHE accelerators, and/or (ii) incur high performance overheads. These gaps highlight the need for systematic evaluation and design of aforementioned countermeasures in the context of FHE accelerators. This will be scope for future work.

E Supplementary Results and Materials

BFV-based CryptoNets Implementation for MNIST Image Classification. As shown in Fig. 14, accuracy significantly decreases when errors are introduced at lower depths, namely for depths less than 8 resulting in accuracy of around $\approx 9\%$. In contrast, errors introduced at the final depth have a reduced effect on accuracy till an error of $5000 \times e$. Similar to CKKS-based CryptoNets, introducing errors at higher multiplicative depths helps to limit output corruption.

Power SCA and FIA Triggering. Our experiments and analysis of the power traces indicates that iNTT exhibits a waveform signature different from other operations in the rescaling process. To that end, we implement the method de-

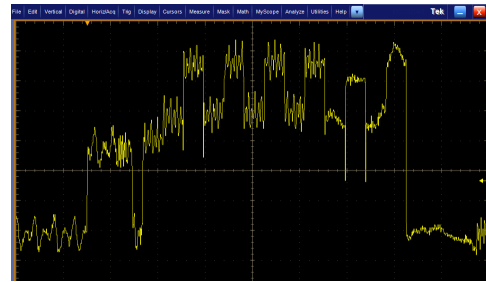


Figure 15: Zoomed-in view of iNTT operation, extracted from the power profile shown in Fig. 12.

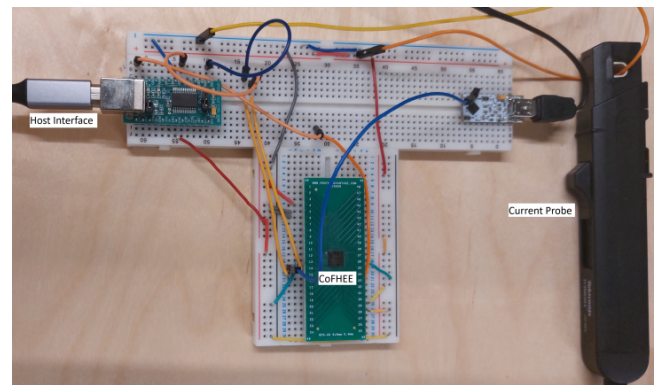


Figure 16: General setup for the CoFHEE ASIC. Note that the current probe shown here is for system-level power SCA; the high-resolution EM probe for localized SCA is not shown.

scribed in [72], employing the *icWaves* transceiver [73] to generate triggers based on real-time waveform pattern matching. Note that this setup enables the loading of a predefined waveform into the transceiver, which subsequently generates a trigger whenever the online power trace corresponds to the stored pattern. Thus, for our experiments, the characteristic waveform of the iNTT operation is preloaded to serve as the trigger reference. More technical details are provided next.

A zoom-in view on the iNTT power trace is shown in Fig. 15. As can be seen, the iNTT operation begins with a multiplication by a constant, followed by $\log_2(N)$ stages of FFT. Recall that, in our experiments, we use a polynomial degree of $N = 2^{13}$. Thus, the waveform reveals 14 distinct regions of elevated power consumption, corresponding to the initial constant multiplication by N^{-1} and the subsequent $\log_2(N)$ FFT stages. As discussed in Sec. 5.4, this characteristic trace can serve as a reference for real-time waveform matching, enabling precise timing for fault injection.

Attack Setup and Equipment. In Fig. 16, we show the general setup for the CoFHEE ASIC, including the host interface and a current probe by Tektronix for system-level power-side channel measurements. Table 4 summarizes the setup for our physical experiments in this work.

The Langer setup [46] used in our work offers high precision for EM fault injection, with probe dimensions as small as

Table 4: Setup and parameters for GlitchFHE experiments.

Device Under Test/Attack: CoFHEE [35]	
Technology	55 nm
Area	$\approx 13 \text{ mm}^2$
Operating Voltage	1.2 V for logic; 3.3 V for IO
Max Frequency	250 MHz
Communication Interface	UART
Max Polynomial Degree (N)	8192
Max Modulus Size ($\log_2 q$)	128
EM FIA Setup: ICI HH500-15 with BPS 202 [46] and ICS 105 Positioning System [74]	
Coil Dimension	500 μm
Pulse Rise Time	< 2 ns
Pulse Repetition Frequency	0.1 Hz to 2000 Hz
Voltage Range	50 V to 500 V
Probe Range (X, Y, Z)	50 x 50 x 50 mm
Positioning Resolution	10 x 10 x 10 μm
Attack Setup	
Chip Frequency	40 MHz
UART Baud Rate	512 kbps
Oscilloscope	Tektronix MSO 6
Current Probe for SCA	Tektronix TCP0030A
EM Probe for SCA	Langer ICR HH 150-27
(X, Y) Coordinates for FIAs	Around 300 μm & 600 μm from top-left corner of die
Probe Height / Z for FIAs	1 μm to 25 μm
Pulse Voltage for EM FIA Probe	> 380 V for single pulse; > 250 V for multiple pulses
Operation Performed	Rescaling

150 μm (we use a 500 μm probe here), along with a positioning system [74] for precise probe positioning. The total cost is moderate, around 15k USD, making it accessible for most research labs. Other more affordable probes, e.g., NewAE CW520 [75] are available for around 4.35k USD, but they compromise on spatial precision for FIA due to their larger probe dimensions, namely 1 mm for the CW520. Further, there is no positioning system readily available for this probe.

In general, as IC technology advances, smaller probe diameters will become more critical to accurately target specific chip regions. Finally, while alternative FIA techniques like clock glitching or voltage glitching [75] may also be used, they face similar challenges for precise fault control.

Attack Guidelines, Steps for Reproducibility. The following steps shall be performed for successful FIA on the CoFHEE ASIC (and other FHE accelerators).

- *Step 1:* Acquire the same device as the one targeted at.
- *Step 2:* Place the EM FIA probe at one of the chip corners, positioned very close to the chip surface.
- *Step 3:* Load data and constants for iNTT operation into the memory and configuration registers.
- *Step 4:* Perform the iNTT operation.
- *Step 5:* Inject a fault via high-voltage pulse (e.g., > 350V).
- *Step 6:* Read back the result from memory and check if the data is corrupted.
- *Step 7:* Systematically position the probe across the X, Y coordinates of the chip surface. Repeat Steps 3–6 until some

Table 5: Nr. of bytes corrupted for 10 back-to-back pulses. For each data point, pulse injection is repeated multiple times, and the majority outcome is recorded.

Pulse Voltage (V)	Pulse Frequency (Hz)					
	10	50	100	500	1,000	10,000 20,000
200	0	0	0	0	0	0
220	0	0	0	0	0	0
240	0	0	0	0	0	0
250	1	0	1	1	1	0
260	0	1	1	1	0	1
280	2	1	1	1	2	1
300	2	2	2	2	2	1
350	2	2	3	2	2	2
400	2	2	2	2	2	2

fault(s) is/are induced. A successful fault should corrupt only a limited memory region (e.g., <100 locations).

- *Step 8:* Fine-tune parameters such as probe position (X, Y, Z) and voltage until faults can be reliably injected into only a few memory cells (e.g., <10), with high success probability.
- *Step 9:* Conduct multiple power SCA runs during the iNTT operation and average them to reduce noise.
- *Step 10:* Position the FIA probe on the actual target device, using the learned coordinates.
- *Step 11:* Wait until the iNTT operation is detected/triggered, by automatically comparing real-time power traces with the reference waveform captured in Step 9.
- *Step 12:* Inject EM pulses, using the learned voltage.

FIA with Multiple Pulses. Upon occurrence of the trigger signal (described above), FIAs can be carried out in two ways: one can inject a single pulse during normal operation or—assuming one can also control/halt the clock⁷—one can inject multiple pulses back-to-back, thereby increasing chances for successful faults, as the data is in rest in this scenario.

Our experiments and analysis shows that using multiple pulses enables successful fault injection already at lower voltages, namely starting from 250 V, compared to 380 V needed for single pulses. More importantly, multiple-pulse injection enables better control, making it easier to incur a specific, single-byte fault. Table 5 reports related results. Indeed, single-byte faults are achieved across various settings.

⁷The Langer setup used in our work can generate either single pulses or multiple pulses, the latter at a maximum frequency of 20 kHz, i.e., there is a gap of at least 50 μs between two EM pulses. However, any target chip's clock typically runs at MHz or even GHz ranges, i.e., by the time a second EM pulse arrives, the data has already advanced, e.g., into the NTT domain. As discussed, FIAs in the NTT domain can result in corruption of decrypted data with noticeably high MSE. Thus, multiple-pulse EM injection is only practical if the attacker can also control/halt the clock immediately after the iNTT operation, or if the FIA setup can inject pulses with a frequency matching the chip operation. For our experiments, we apply the former.