SASTA: Ambushing Hybrid Homomorphic Encryption Schemes with a Single Fault

Aikata Aikata[1], Dhiman Saha[2] and Sujoy Sinha Roy[1]

[1] University of Technology Graz, Austria
{aikata,sujoy.sinharoy}@iaik.tugraz.at
[2] Indian Institute of Technology Bhilai, India
dhiman@iitbhilai.ac.in

Abstract. The rising tide of data breaches targeting large data storage centres, and servers has raised serious privacy and security concerns. Homomorphic Encryption schemes offer an effective defence against such attacks, but their adoption is hindered by substantial computational and communication overhead, both on the server and client sides. This challenge led to the development of Hybrid Homomorphic Encryption (HHE) schemes to reduce the cost of client-side computation and communication. Despite the existence of a multitude of HHE schemes in the literature, their security analysis is still in its infancy, especially in the context of physical attacks like Differential Fault Analysis (DFA). This work aims to address this critical gap for HHE schemes defined over prime fields ($\mathbb{F}_p - \mathsf{HHE}$) by introducing, implementing and validating SASTA, the *first* DFA on $\mathbb{F}_p - \mathsf{HHE}$ and the first nonce-respecting FA over any HHE scheme.

In this pursuit, we introduce a new nonce-respecting fault model (all current fault attacks on HHE schemes require a nonce-reuse), which leads to a unique attack that completely exploits both the asymmetric and symmetric facets of HHE. We target $\mathbb{F}_p - \mathsf{HHE}$ schemes as they offer support for integer or real arithmetic, enabling more versatile applications, like machine learning, and better performance. The fault model benefits from what we call the *mirror-effect*, which allows the attack to work both on the client and the server. Our analysis reveals a significant vulnerability: *a single fault* within the Keccak permutation, employed as an extendable output function, results in complete key recovery for the Pasta HHE scheme. Moreover, this vulnerability extends to other HHE schemes, including Rasta, Masta, and Hera, amplifying the scope and impact of SASTA.

For experimental validation, we mount an actual fault attack using ChipWhisperer-Lite board on the Keccak permutation. Following this, we also discuss the conventional countermeasures to defend against SASTA. Overall, SASTA constitutes the first nonce-respecting FA of HHE that offers new insights into how server-side or client-side computations can be manipulated for $\mathbb{F}_p - \mathsf{HHE}$ schemes to recover the entire key with just a single fault. This work reaffirms the orthogonality of convenience and attack vulnerability and should contribute to the landscape of future HHE schemes.

Keywords: Hybrid Homomorphic Encryption · Differential Fault Attack · Pasta · Rasta · Masta · Hera · Rubato · Single Fault

1 Introduction

The world is witnessing a concerning surge in daily data breaches, typically directed at large data storage centres and servers. These breaches allow attackers to access vast amounts of private information. This is because, even though the data in communication is encrypted, the servers decrypt it for computation or storage. This allows not only the server to see

the client's data but also attackers who can mount data breach attacks. To address this growing security and privacy concern, there is a pressing need for privacy-preserving data storage and computation solutions.

Homomorphic Encryption (HE) schemes [Gen09, CKKS17, BGV11, CGGI20, FV12, Bra12] offer such privacy preservation capabilities. However, their widespread adoption is inhibited due to significant computational and communication overhead. This is because the FHE schemes transform data for processing into substantially larger polynomials and rely on costly polynomial arithmetic for computations. The transmission of these large polynomials incurs huge communication expenses. While server-side computations can be expedited through dedicated hardware accelerators [MAK+23, GVBP+22, FSK+21, KKK+22, AMK+23], the same approach is economically expensive for client-side applications due to the high cost of such accelerators.

To ease the computational and communication burdens on clients and ensure that privacy does not come at a high cost, HHE schemes [DEG+18, DGH+23, HKC+20, CHK+21, HKL+22] have been developed. They are symmetric key schemes with low multiplicative depth, often employing stream ciphers. Initially, these schemes supported operations on boolean data (e.g., Rasta [DEG+18], Flip [MJSC16], Filip [MCJS19], Kreyvium [CCF+18], and AgRasta [DEG+18]), but they have evolved to handle data over prime fields $\mathbb{F}_p$, enabling the processing of integers or real numbers. Assessing real-life applications (e.g., Machine Learning) [JVC18, CMdG+21, BBH+22] in $\mathbb{Z}_2$, demands the construction of binary circuits with considerably greater multiplicative depth to achieve integer arithmetic. An additional issue arises as the batching technique (an encoding technique employed for SIMD-like operations) is unavailable in a power-of-two cyclotomic ring required for boolean data, a ring commonly employed in RLWE-based FHE schemes (e.g., BGV, BFV, CKKS). Consequently, HHE schemes that can only operate on boolean data entail a substantial performance decline compared to simply implementing the use case using FHE. Hence, extension to $\mathbb{F}_p$ significantly enhances their utility by catering to a much larger application base and allowing batching. Overall, HHE schemes keep communication packets small and allow fast symmetric-key encryption, hence simplifying computation and communication on the client side.

Despite their promise, all the HHE schemes over prime field, Pasta [DGH+23], Masta [HKC+20], Hera [CHK+21], and Rubato [HKL+22], are relatively new and have not undergone extensive cryptanalysis, especially in the context of physical attacks. Physical attacks refer to practical attacks conducted in real-world scenarios. While they may not directly compromise the schemes from a theoretical standpoint, they can exploit implementational/runtime vulnerabilities. Given that these schemes will be deployed across various embedded platforms, they must be analyzed and protected against side-channel analysis and fault injection attacks. The HHE designs are still evolving to become more robust and fast, and this analysis will help the designers in designing cryptosystems that are both theoretically sound and practically secure.

In this work, we narrow our focus to fault injection attacks, a technique aimed at recovering private cryptographic keys by introducing physical faults into a device. This approach was first demonstrated on RSA-CRT Signatures [BDL97]. Since then, various settings have been proposed to exploit induced faults, such as DFA [BS97, LYK21], Integral FA [SSL15, KGSR23], Internal Differential FA [SC16], Ineffective FA [Cla07], Statistical FA [FJLT13, DEK+18, VZB+22], and Persistent FA [ZFL+22]. Our work specifically investigates the known plaintext or chosen plaintext fault analysis paradigm DFA, which uses fault-free and faulty ciphertext pairs to construct differentials that can help retrieve private and otherwise unknown information, for example, intermediate state values. This knowledge eventually allows the attacker to get more sensitive information-key, by performing a key recovery.

The effectiveness of DFA lies in its ability to achieve key recovery with a minimal number

of faults. The DFA attack model relies on certain assumptions, such as the ability to induce required faults, choose plaintexts or ciphertexts, and, in some cases, nonce-reuse, a value that should inherently remain unique for an execution. If the attacker can additionally let go of the assumptions, such as nonce reuse or chosen plaintext, then the attack setting gets even weaker and increases the feasibility of the attack in real-world scenarios. To date, there have been few DFA studies [RBM21, RKMR23] on Rasta [DEG+18], Kreyvium [CCF+18], and Filip [MCJS19] HHE schemes. These schemes are designed primarily for boolean data and suffer high latency costs when processing integer or real data. In contrast, more recent schemes like Pasta [DGH+23], Masta [HKC+20], Hera [CHK+21], and Rubato [HKL+22], operate over a prime field $\mathbb{F}_p$, and offer practical solution and wider application support. Analysis of these schemes under DFA is unexplored yet much needed due to the vast potential for deployment of privacy-preserving applications.

In this work, we address this gap and hope this comprehensive analysis will enhance the robustness and security of future designs.

Contributions

This work is the *first* effort in exploiting HHE schemes over $\mathbb{F}_p$ using fault analysis, with results tabulated in Table 1. The notable contributions of this work are as follows:

- **First Nonce-respecting Fault Model for HHE:** We introduce a novel fault attack model tailored for HHE schemes. It is based on the standard DFA model and obtains differentials without repeating nonce. This is particularly significant as it relies on *very weak assumptions* and does not compromise any fundamental algorithmic properties, enhancing the impact and feasibility of the attack.
- **SASTA - A unique attack strategy to break Pasta:** We leverage this novel fault model to develop a distinct attack strategy for breaking Pasta [DGH+23]. Our primary target is the Keccak permutation, a common component among all HHE schemes. Note that the attack is not restricted to Keccak and can be applied to any pseudo-random number generator used in future designs.
- **Extension to Multiple HHE Schemes - Masta, Hera, and Rasta:** We extend our analysis to encompass the Masta [HKC+20], Hera [CHK+21], and Rasta [DEG+18] HHE schemes, illustrating how SASTA can be applied to them. Additionally, our research reveals that due to reliance on Gaussian noise in encryption, Rubato [HKL+22] can withstand SASTA at the expense of substantial precision loss and limited use cases. In essence, our study comprehensively analyzes all state-of-the-art HHE schemes over $\mathbb{F}_p$ and an $\mathbb{F}_2$ scheme Rasta, demonstrating scalability of the attack model .
- **Real-world Realization of SASTA with a Single Fault:** We demonstrate our attack on Keccak using the ChipWhispererLite CW1173 board[1]. Remarkably, this attack requires just a *single known fault* within the Keccak permutation, resulting in complete key recovery. The attack surface is exploitable not only on the client side but also on the server side, significantly amplifying the vulnerability of the system.
- **Countermeasure Analysis:** We finally present a discussion on conventional DFA countermeasures for resistance against SASTA. An analysis of how masking will not be an effective countermeasure is also furnished.

Organization: We introduce the utility and design of HHE schemes in Section 2. Section 3 presents the detailed research gap that is addressed in this work. Section 4 explains our

[1]The code for Keccak fault injection and unique key recovery of Pasta, Masta, Hera, and Rasta will be made available for the Artifacts review.

Table 1: Summary of Fault attacks on HHE and contributions of SASTA for unique key-recovery. The time reported is the average time for key recovery once a faulty and correct keystream is obtained.

Target Scheme	Security	Field	Nonce Repetition	#Faults	Time	Reference
Flip_{530}†	80-bit	$\mathbb{Z}_2$	Yes	1	39hr	[RBM21]
Kreyvium†	128-bit	$\mathbb{Z}_2$	Yes	3	5min	[RBM21]
Filip_{430}‡	80-bit	$\mathbb{Z}_2$	Yes	1	751hr	[RKMR23]
Rasta-6‡	80-bit	$\mathbb{Z}_2$	Yes	1	96hr	[RKMR23]
Rasta-5	128-bit	$\mathbb{Z}_2$	No	1	0.1s	Subsection 5.4
Rasta-6	128-bit	$\mathbb{Z}_2$	No	1	0.12s	Subsection 5.4
Pasta-3	128-bit	$\mathbb{F}_p$	No	1	0.5s	Section 4
Pasta-4	128-bit	$\mathbb{F}_p$	No	1	0.21s	Section 4
Masta-4	128-bit	$\mathbb{F}_p$	No	1	0.09s	Subsection 5.1
Masta-5	128-bit	$\mathbb{F}_p$	No	1	0.07s	Subsection 5.1
Hera-5	128-bit	$\mathbb{F}_p$	No	1	0.005s	Subsection 5.2

† Results obtained on processor with 2.00 GHz clock and 4 GB RAM.
‡ Results obtained on processor with 2.80 GHz clock and 16 GB RAM.
Our results are obtained on processor with 2.00 GHz clock and 4 GB RAM.

attack model and strategy, and we see how it can lead to full key recovery for Pasta. We further analyze this attack strategy for other HHE schemes Masta, Hera, Rubato, and Rasta in Section 5. In Section 6, we discuss how we experimentally inject a fault. Next, we analyze the feasibility and cost of conventional countermeasures in Section 7. Section 8 presents a discussion on future scope and concludes the work.

2 Background

This section starts with a quick overview of homomorphic encryption and why we need HHE schemes. Then, we discuss the design of target HHE schemes and take a closer look at some of the mathematical methods used in the design and implementation, like the 'baby-step giant-step' technique for multiplying matrices and vectors, the Keccak function used for generating pseudo-random numbers, and the Number Theoretic Transform (NTT).
Notations: $\mathbb{Z}_2$ refers to boolean values or integers modulo 2. Similarly, $\mathbb{Z}_p$ refers to integers modulo p. Numbers in $\mathbb{R}$ are the real values. $\mathbb{F}_p$ is used to denote prime fields for a prime modulus p. We will denote numbers (integers and real) as small letters (e.g., x), vectors of numbers as capital letters (e.g., X)(except message m and ciphertext c), and matrices as bold and capital letters (e.g., $\mathbf{M}$). The subscripts X_i are used for naming different matrices or vectors, and the superscripts X^i are used for indexing coefficients in a vector or rows in a matrix. Throughout the paper, variables with apostrophe sign (e.g., $c', C', \mathbf{M}'$) refer to faulty values resulting from fault injection while generating the actual values (e.g., $c, C, \mathbf{M}$). The functions are denoted using typewriter font (e.g., `rot`, $\texttt{A}_i$).

2.1 Hybrid Homomorphic Encryption

Before we delve into the specifics of 'Hybrid' Homomorphic Encryption, let us first understand Homomorphic Encryption. It is one of the four pioneering Privacy Enhancing Techniques, including Trusted Execution Environments, Multi-Party Computation, and Zero-Knowledge Proofs. The first FHE scheme was introduced by Gentry in 2009

[Gen09], and since then, several FHE schemes have been introduced by researchers, such as BGV [BGV11], BFV [FV12, Bra12], CKKS [CKKS17, CHK+18], and TFHE [CGGI20]. These schemes enable computations on encrypted data, making tasks like machine learning possible while preserving privacy. This unique capability of privacy-preserving data storage and computation has made FHE 'the holy grail of cryptography' [Mic10].

Most FHE schemes rely on complex mathematical problems like Learning With Errors (LWE) or its faster polynomial ring variant, Ring Learning With Errors (RLWE), to ensure security and homomorphic computation. However, a significant drawback is that they transform plaintext data into much larger polynomials when homomorphically encrypted, resulting in substantial communication and computational overhead, often ranging from $10,000\times$ to $100,000\times$ [JLK+21]. Transmitting these large ciphertexts and performing operations on them is quite expensive. To address this challenge, researchers have developed specialized hardware accelerators [MAK+23, GVBP+22, FSK+21, KKK+22, AMK+23], to accelerate computation. These accelerators can significantly reduce computation costs, but the issue of communication overhead remains a significant bottleneck in practical implementations. These accelerators are also designed mainly for operations on the server side as they are too expensive for the client side. Hence, clients continue to suffer from both *computational* and *communication* overhead.

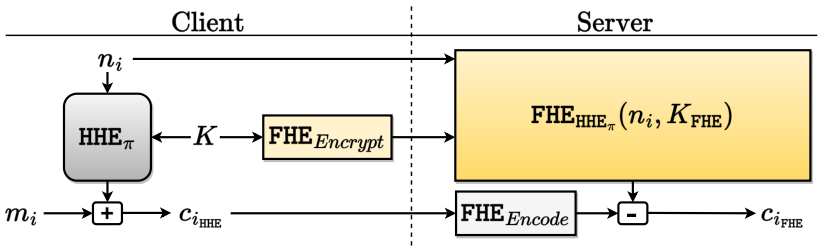

Figure 1: Workflow of HHE schemes is illustrated here. The homomorphically encrypted key ($K_{\mathtt{FHE}}$) is communicated only once. After this, multiple small ciphertexts ($c_{i_{\mathtt{HHE}}}$) encrypted using HHE are sent along with nonce. The server performs a 'homomorphic HHE decryption' (HHD) and obtains a homomorphically encrypted ciphertext $c_{i_{\mathtt{FHE}}}$.

This led to the development of HHE schemes, essentially symmetric key-based encryption methods that can be decrypted in a homomorphic manner. The transition of ciphertext from HHE to FHE is illustrated in Figure 1. The stepwise HHE-FHE protocol is explained as follows.

- The client has two types of keys, the key K for HHE and K_2 for FHE. While K is a symmetric key used for HHE encryption and decryption, K_2 is an asymmetric key consisting of a private (sk) and public key (pk). The client needs to protect both K and $K_{2_{sk}}$ because knowledge of $K_{2_{sk}}$ can leak K, and this, in turn, results in the attacker gaining access to all messages m_i as well as the results sent by the server.

- In the FHE-HHE protocol, the client uses the public key $K_{2_{pk}}$ to encrypt K using $\mathtt{FHE}_{Encrypt}$, and sends this to the server at the beginning itself. After this, whenever clients need to store or process data on the cloud, they use HHE Encryption ($\mathtt{HHE}_\pi$) to encrypt the message blocks m_i using K and send the resultant ciphertext $c_{i_{\mathtt{HHE}}}$ to the server along with the public Nonce n_i.

- The server uses the encrypted key $\mathrm{K}_{\mathtt{FHE}}$ along with the Nonce n_i to evaluate the HHE decryption (HHD) circuit homomorphically ($\mathtt{FHE}_{\mathtt{HHE}_\pi}$). This results in a ciphertext ($c_{i_{\mathtt{FHE}}}$) which is homomorphically encrypted under key K_2. This can now be stored on the server or processed as desired by the client.

- The client can retrieve the ciphertext and use $K_{2_{sk}}$ to decrypt the result. This protocol is followed by all HHE schemes, and its ciphertext transformation capability from one encryption form to another (e.g., $c_{i_{\texttt{HHE}}}$ to $c_{i_{\texttt{FHE}}}$) also leads to its utility in other applications like transciphering.

Initially, HHE schemes were designed to work with binary data ($\mathbb{Z}_2$), such as Rasta [DEG+18], Flip [MJSC16], Filip [MCJS19], Kreyvium [CCF+18], and AgRasta [DEG+18]. However, in response to the growing need for better performance and application support, they have evolved into schemes like Masta [HKC+20], Pasta [DGH+23], Hera [CHK+21], and Rubato [HKL+22] that can directly operate over prime fields $\mathbb{F}_p$, enabling operations on real or integer plaintexts. Besides offering cost-effective communication and efficient (client-side) encryption, as mentioned above, these schemes also find applications in *transciphering*, allowing servers to transform ciphertexts from one FHE scheme to another without decryption.

Next, we will briefly delve into the design of HHE schemes.

2.2 HHE scheme design overview

We have chosen Pasta [DGH+23] for explaining the designs of HHE schemes due to its comprehensive support for various operations. Other designs can be viewed as adaptations or variations of Pasta and will be discussed afterwards.

In the context of HHE, the variables can be categorized into two types: public and private. As illustrated in Figure 1 and Figure 2, both the nonce (n) and the counter (i) are considered public data, as they are known to both the client and the server. In contrast, the key (K) is private and exclusively known to the client. For HHD, this key is encrypted and securely transmitted to the server as $K_{\texttt{FHE}}$ (done only once in the beginning). Hence, both K and the client's message (m_i) remain concealed from the server.

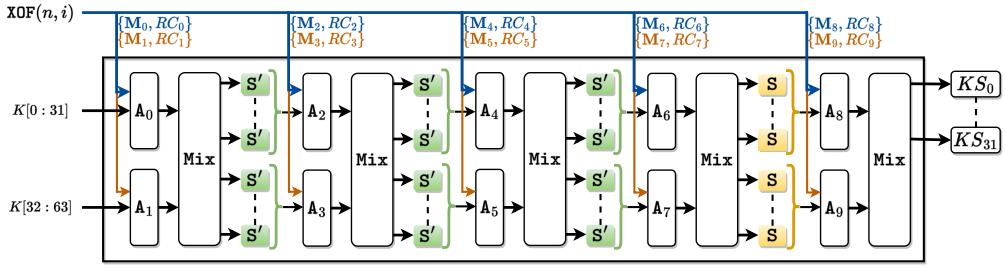

Figure 2: The Pasta-4 permutation (π) takes as input the key (K), nonce (n), and counter (i), and ultimately generates the truncated result- key stream (KS).

With the public and private variables clearly defined, we now describe the Pasta HHE scheme. This scheme operates as a stream cipher and comprises two variants: 3-round Pasta-3 and 4-round Pasta-4. Figure 2 demonstrates the Pasta permutation (π). It is important to note that the operations outside the square box, denoted as `XOF` (extendable-output function), are public. SHAKE128 is used for this. Contrarily, the operations within the box are considered private (key-dependent) and involve either addition or multiplications using modular arithmetic in $\mathbb{Z}_p$. Here, p can be any prime between 16 and 60 bits depending on the specific requirements of the underlying FHE scheme.

The state size ($2t$) varies between the Pasta-3 and Pasta-4 variants of the scheme. Specifically, for Pasta-3, $2t = 128$ coefficients, while for Pasta-4, it is 64. These $2t$ coefficients are divided into two halves, X_L and X_R, and then processed via permutation. The resulting KS is added to the plaintext for encryption and subtracted from the

ciphertext for decryption. The permutation consists of several layers that are applied in each round, described as follows:

- $\mathtt{A}_i$ (Affine Layer): For this layer, an *invertible* matrix $\mathbf{M}_i$ and a round constant vector RC_i are generated using the SHAKE128 `XOF` output. Then, the layer performs $\mathbf{M}_i \cdot X_i + RC_i$ operation, where X_i represents the input state comprising t coefficients.
- `Mix` (Mixing Layer): Following $\mathtt{A}_i$, the two halves of the state are mixed using the `Mix` operation. This operation transforms the state into $(2 \cdot X_L + X_R, 2 \cdot X_R + X_L)$. This step is crucial for spreading values evenly across the two-state halves.
- $\mathtt{S}'/\mathtt{S}$ (S-Box Layer): The next layer involves the S-Box operation. For the final round, the cube S-Box ($\mathtt{S}$) is applied, while for all previous rounds, a Feistel S-Box ($\mathtt{S}'$) is utilized. Both these s-boxes are *invertible*.

$$\mathtt{S}(X^j) = (X^j)^3 \pmod p \quad \forall\ 0 \le j < t$$

$$\mathtt{S}'(X^j) = \begin{cases} X^j \pmod p & \text{if } j = 0, \\ X^j + (X^{j-1})^2 \pmod p & \text{otherwise,} \end{cases}$$

- Truncation Layer (`Trunc`): This is applied at the end (pre-final) and truncates the output to prevent round inversion. It returns the X_L state as the final output.

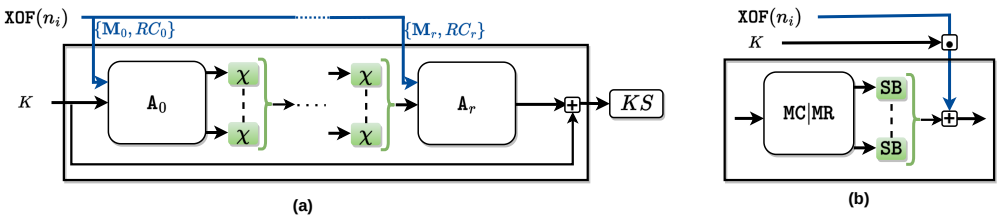

Figure 3: The r-round Masta permutation is illustrated in Figure (a). Figure (b) gives a general idea of Hera and Rubato permutation. The '+' and '·' signs in the boxes refer to modular vector addition and modular vector dot product.

Masta [HKC+20] follows a slightly different approach as shown in Figure 3 (a). It does not split the state into two halves like Pasta and comprises only two primary layers: the affine layer and the S-box layer. The affine layer of Masta is similar to that of Pasta; however, Masta employs χ-S-box (Equation 1). The pre-final step is the most significant distinguishing factor between Pasta and Masta. While Pasta opts for truncation at this stage, Masta employs modular addition with the key. Both Pasta and Masta are versatile and can support operations over $\mathbb{Z}_p$, making them suitable for BGV and BFV. Rasta [DEG+18] is similar to Masta with the only difference being operation over $\mathbb{Z}_2$ and the choice of S-box.

$$\mathtt{S}_\chi(X^j) = \begin{cases} X^j \pmod p & \text{if } j \le 1, \\ X^j + X^{j-1} \cdot X^{j-2} \pmod p & \text{otherwise,} \end{cases} \tag{1}$$

Hera [CHK+21] adopts a distinct approach compared to Pasta and Masta, as shown in Figure 3 (b). It comprises five rounds, all utilizing cube S-boxes. The primary distinguishing feature of Hera is its approach to the affine layer. In this layer, matrix multiplication (`MC|MR`) utilizes constant low-hamming weight matrices, while the addition of round constants is transformed into an add-round key function. For this, the output

of the XOF is multiplied by the encryption key, creating a key-schedule-like effect. After the S-box (SB) operation, every round adds this derived key to the state. Importantly, HERA can be applied to FHE schemes over both integer arithmetic $\mathbb{Z}_p$ (BGV and BFV) and approximate arithmetic $\mathbb{R}, \mathbb{F}_p$ (CKKS).

RUBATO [HKL+22] was developed after PASTA and HERA, utilizing design principles from both schemes (Figure 3 (b)). Specifically tailored for CKKS FHE scheme, RUBATO employs the same design philosophy as HERA. However, it deviates from HERA by employing feistel S-boxes instead of cube S-boxes for all rounds. In its pre-final step, RUBATO utilizes truncation like PASTA and adds Gaussian noise, making the ciphertext dependent on the hard problem of LWE. This, in turn, limits its use case to CKKS and introduces an inherent precision loss in the computation.

We will next understand some of the fundamental building blocks employed by these encryption schemes, which will help comprehend the different attack strategies.

2.3 Baby-step giant-step method

The matrix multiplication required for processing the affine layers is an expensive operation. On the client side, this is not a problem, as data can be fetched and stored as desired. Contrarily, on the server side, data is encoded/encrypted into polynomials, which do not offer much flexibility and consume significant multiplicative depth. To reduce this, baby-step giant-step method [HS18] is used for expediting matrix-vector multiplication on encoded/encrypted data, as shown in Subsection 2.3. The matrix ($\mathbf{M}$) represents encoded plaintext data in this context, and the vector corresponds to encrypted ciphertext (X). t is the vector size, and $t = t_1 \cdot t_2$. The $\mathtt{rot}_j$ function rotates the input vector by j, and the diag function gives rotated diagonals of $\mathbf{M}$.

$$Y = \sum_{k=0}^{t_2-1} \mathtt{rot}_{kt_2} \left(\sum_{j=0}^{t_1-1} \mathtt{diag}_{kt_1+j}(\mathbf{M}) \odot \mathtt{rot}_j(X) \right)$$

This method involves the following crucial steps:

1. *Diagonal Encoding*: The diagonals of the matrix are homomorphically encoded.
2. *Rotation*: Next, these encoded diagonals, which are homomorphic plaintexts, are subject to rotation operations.
3. *Multiplication*: Finally, the rotated diagonals are multiplied with the rotated ciphertext. The resulting ciphertexts undergo a few more rotations and accumulations.

Due to its remarkable efficiency, this method has been used to boost the performance of the expensive bootstrapping procedure in FHE schemes. Hence, the authors in [DGH+23] have proposed to utilize this method for HHD on the server side. This also applies to MASTA. However, HERA, or RUBATO do not use this since they can use more optimized addition-based matrix multiplication due to the low hamming weight of the matrices used in affine layers.

2.4 KECCAK and Number Theoretic Tranform

The KECCAK [PA11] permutation function is renowned for its role as the underlying component of the secure hashing algorithm standard SHA-3. For HHE schemes, KECCAK, in its SHAKE128/SHAKE256 modes, is utilized as XOF, and its inputs are the nonce and counter values. The data generated undergoes a process known as rejection sampling, where it is filtered to ensure that it is smaller than the prime modulus. This output is then used to generate matrices and round constants in schemes like PASTA and MASTA

and obtain the key schedule for Hera and Rubato. Other Pseudo-Random Number Generators (PRNGs), such as AES (as used in [Beu22]), could also be used as alternatives. This is because only the `XOF` property of the scheme is needed, and AES has suitable diffusion property to function as a good `XOF`.

The **Number Theoretic Transform** (NTT) [LN16, Spr] facilitates the conversion of polynomials from coefficient representation to slot representation. With this transformation, polynomial multiplication has a cheap $\mathcal{O}(n \log n)$ complexity, in contrast to the expensive $\mathcal{O}(n^2)$ complexity in the coefficient representation. All ciphertexts and plaintexts utilized on the server side are either stored in NTT form or converted to it for homomorphic multiplications. The NTT transformation can be seen as a matrix-vector multiplication where the input polynomial is the vector, and the matrix comprises powers of roots-of-unity.

Next, we discuss in detail the research gap that we address in this work.

3 The Uncharted Research Gap

Cryptography is a dynamic field, and attackers are constantly evolving their techniques. The iterative process of attacking and proposing countermeasures is essential for maintaining robust encryption solutions. This work delves into the domain of differential fault attacks. Studying DFA attack models is crucial for evaluating and enhancing the security of new and evolving HHE schemes in the embedded context. It provides insights into potential vulnerabilities and leads to the development of better scheme designs or countermeasures to detect, prevent, or mitigate faults intentionally induced by adversaries. Understanding DFA attack models is also essential for meeting security standards and certifications for addressing real-world threats associated with fault injection attacks. In the context of HHE, we observe that very few DFA attacks have been mounted on these schemes, for example, [RKMR23, RBM21] analyzing Rasta [DEG+18], Kreyvium [CCF+18], Flip [MJSC16], and Filip [MCJS19].

The DFA model [TMA11, AKS20] assumes a black box setting, wherein the attacker is restricted to observing public parameters (Nonce), inputs, and outputs during the limited timeframe they have access to the device. Within this scope, the attacker holds the capability to introduce faults into the device's operation through methods like voltage glitches or clock perturbations and then analyze the resulting outputs. The private keys utilized for encryption remain undisclosed to the attacker, preventing them from deciphering the messages transmitted or received by other users via eavesdropping on the communication channel. Nevertheless, if the attacker successfully acquires the private key through the differential constructed from fault injection, the confidentiality of both past and future communications of the device under attack becomes compromised. When the same private key is used by different devices, all of them become compromised by a single fault injection. This is the general setting for a DFA on any symmetric key scheme, and the prior works on DFA often make assumptions such as the ability to choose plaintext or nonce when more than one encryption is required.

Such assumptions help the attacker, but it also makes the attack less likely to succeed in reality. For example, additional precise attacks would be required to ensure that the device does not refresh the nonce (as stated by the protocol). Things can become worse in the context of HHE if the sever and client agree on a common algorithm to generate Nonce on both sides, aiming to minimize communication costs without compromising security, given that the Nonce is a public parameter. In this case, to make sure the Nonce is repeated, an identical attack would be required on both the client and server side, rendering the feasibility of the attack even more impractical. This implementation choice does not affect the integrity of the HHE-FHE protocol but successfully prevents basic DFA attacks on HHE schemes, exposing the limitations of previous DFA efforts [RBM21, RKMR23].

These prior works also differ significantly from our work. The primary distinctions lie

in the targeted schemes. While previous studies focused on schemes over $\mathbb{Z}_2$, our work, for the first time, delves into schemes operating over prime fields (Pasta, Masta, Hera, and Rubato). These schemes hold great promise in practical applications compared to schemes over $\mathbb{Z}_2$. Additionally, prior works treat HHE schemes only as symmetric key schemes and do not respect the nonce-constraint. This makes these attacks somewhat unrealistic. On the contrary, our SASTA approach exploits the HHE framework, eliminating the need for nonce-reuse. Moreover, unlike previous works, we introduce real fault injections. Finally, our research specifically targets schemes and parameters designed to provide 128-bit security, whereas most prior works addressed 80-bit security parameters.

Given the fundamental differences between schemes over boolean data and those over prime fields, there is no known way to apply the existing attacks to schemes over $\mathbb{F}_p$. To the best of our knowledge, no exploration of physical attacks has been conducted on ciphers operating over $\mathbb{F}_p$, which offer the best performance results to date. This is a crucial research gap in the literature. Hence, our work aims to bridge this gap by investigating these HHE schemes under fault injection attacks.

A natural question arises- *how critical is studying such an attack scenario for* HHE *schemes?* The answer to this hinges on the increasing importance of homomorphic encryption in privacy-critical applications like secure data storage and healthcare diagnostics. In these scenarios, preserving the confidentiality of patient data is paramount. An attack that can leak the healthcare institution's secret key could jeopardize the privacy of all associated patients. Furthermore, these attacks could be easily executed by malicious insiders, such as nurses or operators within the healthcare facility.

Our work leverages these practical scenarios to illustrate how they can lead to significantly strong security breaches. As HHE schemes continue to evolve, it is essential to account for physical attacks in their design process. Doing so will not only enhance their inherent robustness but also result in cost-effective countermeasures.

4 SASTA: The Differential Fault Attack on HHE Schemes

Differential fault attacks [TMA11, AKS20] aim to analyze the difference between fault-free and faulty computations. In this scenario, the objective of the attacker is to pinpoint an optimal fault location, such that the result after successful fault injection at this location can be used to reveal private information, often the encryption or decryption key. The beauty of fault attacks relies on two questions: i) how simple is it to inject the fault, and ii) how critical is the attack? The goal being identifying the simplest possible attack scenario that can cause the most significant damage. We will address these questions in the upcoming sections and shed light on the need for more robust HHE designs.

4.1 Defining the Fault Model battlefield

The traditional differential fault attack model relies on the following basic assumptions about the attacker:

- *Fault Injection Capability*: The attacker possesses the ability to inject faults during cryptographic computations.
- *Access to Plaintext-Ciphertext*: The attacker has access to plaintext-ciphertext pairs (e.g., via communication eavesdropping).
- *Few Faults*: The number of faults must be small to prevent any damage to the device.
- *Nonce-Reusability*: The attacker can reset and restart the cryptographic execution with the same public parameters, allowing for nonce and plaintext repetition.

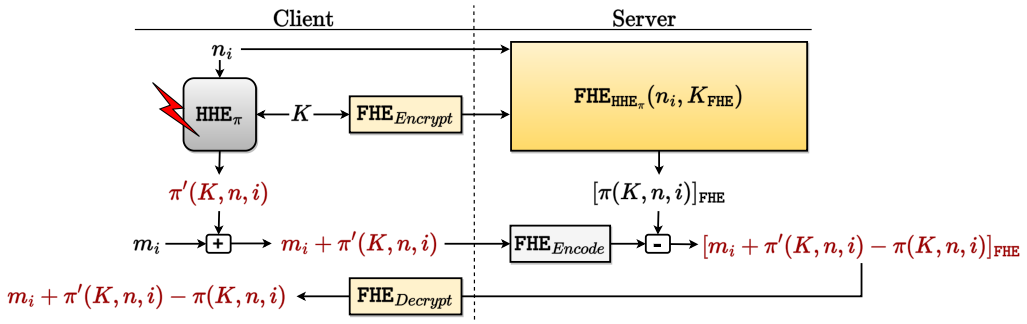

Figure 4: Proposed fault model for differential fault analysis of HHE schemes

While the first three assumptions remain realistic and attainable, the last assumption raises significant concerns. It is based on the idea that an attacker can manipulate aspects of the encryption process (nonce) to make it produce the same ciphertext for the same plaintext. However, the nonce is supposed to change for each operation, ensuring that even identical plaintexts yield distinct ciphertexts. An implementation of HHE schemes will have this built-in feature, and an attacker must execute *precise multiple* fault injections to bypass this. Moreover, most schemes rely on the nonce-respecting setting to prove their security. If an attacker violates that, then the schemes may already be vulnerable to theoretical attacks. Hence, the nonce-reuse scenario is perceived as being artificial. Thanks to SASTA, we do not need this assumption and the fault model outlined below does not require a nonce-reuse. In fact, one of the main contributions of this work is to showcase that a fault-induced differential is achievable like classical DFA without nonce-replay by just leveraging the way HHE schemes work both on the server and client sides.

Given the inability of the attacker to replay the public parameters, they are restricted to a single faulty plaintext-ciphertext pair per public parameter. To further enhance the applicability of our attack, we only consider the case of a single fault during an execution to mount the attack.

We will now discuss how a differential is obtained for this DFA fault model in the context of HHE, as depicted in Figure 4. It is worth noting that the classical DFA setting that requires nonce-replay essentially fails in the nonce-respecting scenario and forms the basis of the new framework developed below, which allows to mount DFA without nonce-reuse exploiting the HHE setting.

Lemma 1 (**Expression of Faulty Ciphertext**). *If a fault occurs during the client-side* Pasta-π *permutation, the resulting ciphertext* c *with* t *coefficients after encryption can be expressed as follows:*

$$c'^j = m^j + \pi'(K,n,i)^j \pmod{p} \quad \forall\ 0 \leq j < t$$

Proof. In the context of HHE, the ciphertext c^j is a function of the original message m^j and the permutation $\pi(K,n,i)^j$ applied during encryption. When a fault occurs during the permutation process, this faulty ciphertext is represented as the sum of the original message and the output of the faulty permutation. □

Lemma 2 (**Computation of Faulty Message**). *In the event of a fault during client-side* Pasta-π *computation, the server-side* HHD *results in a faulty message* m'*, calculated as follows:*

$$m'^j = m^j + \pi'(K,n,i)^j - \pi(K,n,i)^j \pmod{p}\ \forall\ 0 \leq j < t$$

Proof. The server, upon receiving the ciphertext c, performs HHD, which is essentially the subtraction of the encrypted permutation $\pi(K, n, i)^j$ from c ([DGH+23] Section 7):

$$m^j = c^j - \pi(K, n, i)^j \pmod{p} \; \forall \; 0 \leq j < t$$

When a fault occurs during the client-side permutation, it results in a deviation from the actual computation. The server-side HHD uses the faulty ciphertext and the fault-free permutation $\pi(K, n, i)^j$. This results in the computed message being faulty m', differing from the original message m due to the fault-induced difference in the permutations.

$$m'^j = c'^j - \pi(K, n, i)^j \pmod{p} \; \forall \; 0 \leq j < t$$

□

Note that the server has no way to know this, and *this is only perceived by the client* when they homomorphically decrypt the result from the server.

Theorem 1 (**Fault induced differential without nonce-reuse**). *When a fault occurs in the* Pasta-π *permutation during the (client-side) encryption, then due to server-side* HHD*, the fault induced differential is as given below.*

$$\Delta\pi = \pi'(K, n, i) - \pi(K, n, i) \pmod{p}$$

Proof. Using Lemma 1, we get a faulty ciphertext c' when a fault is injected in the Pasta-π permutation during the encryption. Lemma 2 shows that the server-side HHD of c' yields a homomorphically encrypted but faulty message m' distinct from the original message m. Since the original message is known to the attacker, the differential ($\Delta\pi$) can be obtained by simply subtracting the known message m from m'.

$$m' - m = \pi'(K, n, i) - \pi(K, n, i) = \Delta\pi \pmod{p}$$

□

Corollary 1 (**The mirror effect**). *The same fault model can be used on the server side during* HHD *to obtain the differential:*

$$\Delta\pi = \pi(K, n, i) - \pi'(K, n, i) \pmod{p}$$

Proof. For all HHE schemes that are stream ciphers there is a computation mirror effect on the client and server side. Hence, without any loss of generality, the same fault model can be used by the adversary, who could inject the faults on the server side. □

4.2 Unique Differential Acquisition in SASTA Fault Model

The SASTA fault attack model diverges significantly from traditional fault models in both symmetric and asymmetric cryptography. In typical scenarios, where two parties communicate for secure key exchange or encrypted data exchange, attackers can realistically target only one party and access its results with no access to the other end. This necessitates assumptions like nonce repetition to create a pair of original and faulty ciphertexts for differential analysis. In the case of HHE schemes, the HHE encryption happens on the client side, and the HHE decryption is performed by the server. Despite HHE encryption and decryption occurring at different locations (client and server), the HHE decrypted result is eventually sent back to the client after storage/processing, as the client uses the server as a service provider and can request the data stored or results of computations.

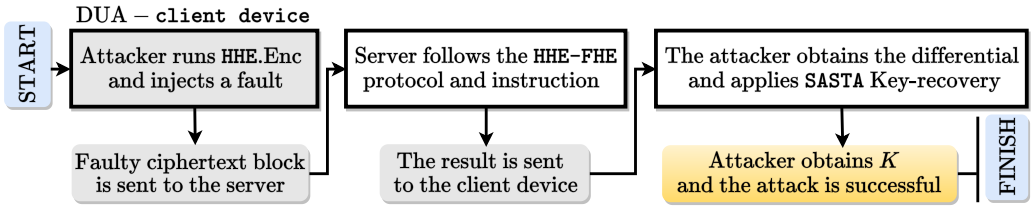

Figure 5: A flow diagram illustrating the full attack step-wise overview.

This unique characteristic allows attackers to obtain differentials without the need for repeating public parameters, making the attack setting highly realistic and the attack brutal. The client queries the server to perform desired computations. Hence, the computation that the server has to do is controlled by the client/attacker and can be something as simple as a basic addition operation. As an example of SASTA on the client-side, imagine an attacker posing as a nurse entrusted with sending patient information to the server. The attacker injects a single fault during the PASTA encryption process, receives the data/result from the server, and unveils the secret key of the hospital. Subsequently, the attacker gains access to all past and future patient information by simply monitoring the communication channel between the server and the client-device. Hence leading to a breach of privacy for all the patients. Similarly, in the context of a smart home security system, an attacker posing as a device technician could inject a fault during the encryption of home surveillance data which is then stored in the cloud. This could lead to unauthorized access to live camera feeds, compromising the privacy and security of residents within the smart home environment.

4.3 Full Attack Overview

Before delving into these specifics of how such a fault is induced and key recovery is performed, we first provide an attack overview. All traditional DFA attacks require at least two ciphertext communications with the chosen nonce or plaintext assumptions. This is so that correct and faulty ciphertext can be obtained under the same plaintext and public parameter setting to form a differential. However, SASTA eliminates the need for such assumptions by only requiring a single faulty query to the server. A step-by-step overview of the full attack (shown in Figure 4.3) is as follows:

- *Execution of* HHE *Encryption and Fault Introduction*: The HHE encryption is executed and a fault is induced during the generation of the last vector/matrix using XOF. This results in the generation of erroneous outputs.
- *Observation of Differences*: The attacker carefully observes the differential obtained in the output returned by the server. This differential is formed due to the difference in the keystreams, as the client sent a ciphertext which is encrypted using an erroneous keystream, whereas the server performed HHD using the correct keystream.
- *Extraction of Sensitive Information*: The attacker uses the information gained from the fault analysis to extract the private key K stored in the device used for the HHE encryption and decryption. The key-recovery method is explained in the next section (Subsection 4.4). This information can then be leveraged to compromise the privacy of users of the attacked device.

In some cases, the attacker may reiterate this process by introducing additional faults or refining the fault injection strategy to enhance the success of the attack. However, in the case of SASTA no additional iteration is required.

In the following subsections, we will explore how to induce this fault to leverage it for key recovery.

4.4 The DFA ambush strategy

Our objective is to employ a single fault that can lead to sufficient diffusion while ensuring that the faulty result is easily exploitable. We will discuss how to achieve this in the context of PASTA and, in the next section, illustrate how it can be applied to other schemes.

Several opportunities exist for inducing a fault such that it diffuses widely throughout the PASTA-4 permutation (Figure 2). For instance, introducing a fault before the final S operation will result in diffusion across the entire state due to the subsequent matrix multiplications ($\mathtt{A}_{8/9}$) and Mix steps. Now, using the non-linear properties of S, equations can be formed to guess the state before the S operation. The possibilities will then be reduced based on the obtained differential. However, this differential alone will not lead to a unique key recovery; it would necessitate multiple such faults to eliminate potential key guesses. Hence, while this approach is feasible, we conclude that it may not be the most optimal solution. Therefore, we seek alternate fault injection possibilities.

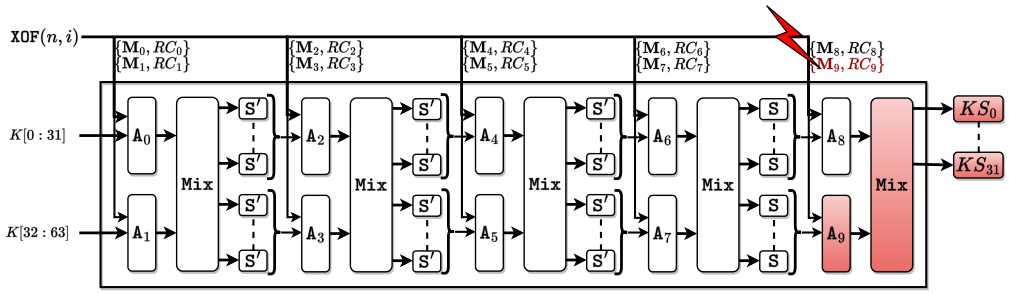

Figure 6: The attack strategy and fault injection point for PASTA-4 permutation

Here, we reiterate our aim to inject a single fault that leads to complete key recovery. Keeping this goal in mind, let us explore the impact of introducing a fault in the matrix to be multiplied during step $\mathtt{A}_9$, as depicted in Figure 6. If we denote the input state after the last S operation as X_8 and X_9, the correct result is as follows:

$$KS = \mathtt{Trunc}(\mathtt{Mix}(\mathtt{A}_8(X_8), \mathtt{A}_9(X_9))) \tag{2}$$

$$KS = 2 \cdot (\mathbf{M}_8 \cdot X_8 + RC_8) + (\mathbf{M}_9 \cdot X_9 + RC_9) \tag{3}$$

The Trunc operation prevents easy round inversion for key recovery. After fault injection, we obtain the following:

$$KS' = 2 \cdot (\mathbf{M}_8 \cdot X_8 + RC_8) + (\mathbf{M}'_9 \cdot X_9 + RC_9) \tag{4}$$

$$\Delta KS = \Delta\mathbf{M}_9 \cdot X_9 \tag{5}$$

$$X_9 = \Delta\mathbf{M}_9^{-1} \cdot \Delta KS \tag{6}$$

Lemma 3. *Uniquely Recovering X_8 and X_9 implies full key-recovery of key K.*

Proof. The permutation consists of matrix multiplications, which involve invertible matrices, and the remaining linear operations are inherently invertible. Therefore, there is a one-to-one mapping between the initial input state K, public variables n, i, and intermediate state X_8, X_9. This mapping is based on the fixed round operations in between and the public XOF outputs $(\mathbf{M}_i, RC_i)$. If we can obtain X_8, X_9 uniquely, then this will lead to a unique key recovery because all the operations in the permutation are invertible. □

Lemma 4. *ΔM_9 is known and invertible.*

Proof. $\Delta\mathbf{M}_9$ is the difference caused by fault injection in the matrix $\mathbf{M}_9$, and its value is known due to known-fault position. For ensuring invertibility with high probability [KKW08], we show in Subsection 4.5 how we choose appropriate fault injection points. □

Theorem 2 (**Unique Key-recovery**). *The* SASTA *attack strategy leads to a unique key-recovery for* PASTA.

Proof. To understand this, let us categorize the variables in the above equations into two types- known and unknown. The only unknown variables are the states halves X_8 and X_9. We know KS', ΔKS, and consequently KS' as these are the results of faulty client-side permutation ($\pi'(K, n, i)$) and the differential obtained from the server output ($\Delta\pi$). The matrices $\mathbf{M}_8, \mathbf{M}_9$ and round constants RC_8, RC_9 are generated using the `XOF`. The inputs to `XOF` are nonce and counter, which are public. Hence, these variables are also known.

If we can use fault to recover private information X_8, X_9, it would lead to a complete key recovery, as shown in Lemma 3. Truncation (Equation 3) prevented this inversion by revealing only half the state. Now, using a fault attack, we will recover the remaining half. To do this, we will recover X_9 using Equation 6, given that $\Delta\mathbf{M}_9$ is known and invertible (4). We can use this information to recover X_8 from Equation 4, mitigating the truncation. Thus leading to unique key recovery. □

Next, we will explore determining the optimal location for introducing the fault.

4.5 Locating the Fault Injection Points

Ensuring $\Delta\mathbf{M}_9$ is invertible after a fault is crucial for determining the appropriate fault injection point (FIP). However, before delving into the specifics of how this is ensured, let us first understand how the original invertible matrices ($\mathbf{M}_i$) are generated. The steps for generating the invertible matrices of PASTA-π are as follows:

- Generate a vector of t numbers using SHAKE128 `XOF`. This constitutes the first row of the matrix $\mathbf{M}_i^0 = [\alpha_0, \alpha_1, \cdots, \alpha_{t-1}]$.
- Next, use this row to generate the remaining rows using Equation 7 [GPP11, GPPR11]. This ensures that the matrix is invertible

$$\mathbf{M}_i^{j+1} = \mathbf{M}_i^j \cdot \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ 0 & 0 & 0 & \cdots & 1 \\ \alpha_0 & \alpha_1 & \alpha_2 & \cdots & \alpha_{t-1} \end{bmatrix} \quad \forall\ 1 \leq j < t \tag{7}$$

Before identifying the precise attack locations let us discuss the devices under potential attack. For a successful fault injection exploit, the attacker needs access to plaintext, as well as the ciphertext exchanged by the client's device. Therefore, a brief period of eavesdropping on the client device's input and output ports becomes necessary. It is crucial to note that the private key is stored within the device and is not part of the input. Subsequently, the attacker can initiate the attack on either the client side or the server side. We argue that although attacking the client's device is more realistic and is generally the target of all prior works, we cannot completely rule out the possibilities of the attack being induced on the server side by a malicious insider. Hence, it is imperative to inform the community about the potential of such an attack, acknowledging that its feasibility remains an open debate (akin to the case of Hardware Trojans).

FIP Case 1: Server-side - Encoding(NTT) of Matrix diagonals

Let us examine the ideal scenario where a fault (non-zero by definition) fully diffuses only to the diagonal of the matrix $\mathbf{M}_9$. This ensures $\Delta\mathbf{M}_9$ is invertible, as it is a diagonal matrix with non-zero elements in the diagonal. However, achieving this level of diffusion with just a single fault on the client side is challenging. Here, the server-side computation offers a remarkable opportunity, thanks to the use of baby-step giant-step based matrix-vector multiplication method (discussed in Subsection 2.3). This method encodes the diagonals of matrix $\mathbf{M}_9$ in NTT form (discussed in Subsection 2.4). These encoded diagonals are then utilized for multiplication.

When the fault is introduced on data in NTT form, it diffuses throughout the entire plaintext after the Inverse NTT operation. This is because of the matrix multiplication used for reversing the transform. Since our plaintext is a diagonal, the fault spreads to the whole diagonal. For practically inducing this known fault, an instruction skip fault during the NTT transform is sufficient, as the diagonals are public. Such a fault can be induced with high precision as shown in [DRPR19, MDP+20, BFP19, BH22]. However, attacking the server-side computation can be challenging due to limited opportunities and constraints. Thus, our next step is to explore potential strategies for achieving our objective on the client side.

FIP Case 2: Client and Server-side - `XOF`

On the client side, injecting a single fault that exclusively diffuses to the diagonal is very difficult as client side computation does not use baby-step giant-step. Hence, we consider the case when the entire matrix is affected by the fault. To induce such a fault, we identify the Keccak *permutation* (`XOF`) as an attractive option. A fault in the `XOF` function, just before it generates $\mathbf{M}_9$, would lead to a full matrix diffusion since the initial vector $\mathbf{M}_9^0$ would be faulty. This exploits Keccak's diffusion property against HHE. Observe that this fault model is not instruction-specific. The sole requirement is knowing which instruction is skipped so that the attacker can know the value of $\Delta\mathbf{M}_9$.

The only potential issue that remains is that the difference matrix may not always be invertible. Studies, such as [Rud06, DS01, KKW08], indicate that the probability of random matrices being invertible is close to one ($\prod_{k=1}^{n}(1-q^{k-1-n}) < 1 - \frac{1}{q}$, where n is for matrix of dimension $n \times n$). In our case, where the initial vectors have no linear dependency, we have experimentally verified a similar high probability of $\Delta\mathbf{M}_9$ being invertible. Hence, the attack will work in almost all cases. Note that this technique is *not limited* to Keccak and can also be extended to the use of other PRNG, like AES.

Next, we assess the risks associated with SASTA.

4.6 Evaluating the Damage: Risks and Implications

We show this attack in the case of Pasta-4; however, without any loss of generality, the same can be applied to Pasta-3 as well. After a successful fault is injected (demonstrated in Section 6), the key recovery takes a few milliseconds on a CPU. The strength of our proposed attack- SASTA lies in the weakness of the attack model. SASTA demands only a single non-specific fault injection and leads to full key recovery. As a consequence, the security of both past and future HHE encrypted ciphertexts is jeopardized.

Even if a fault detection method were to detect and alter the key for subsequent iterations, it would leave all previously encrypted messages vulnerable. Depending on the sensitivity of the data involved, this breach of privacy could have severe and far-reaching consequences. Consequently, safeguarding against such attacks requires a dual approach, involving not only fault detection but also effective mitigation strategies to ensure the privacy and integrity of data at all costs.

In summary, the attack model and strategy presented here are generic and have a huge impact on the security requirements of PASTA. In the following sections, we will explore how this strategy can be extended to other HHE schemes- MASTA [HKC+20], HERA [CHK+21], RUBATO [HKL+22], and RASTA [DEG+18].

5 Broadening the attack horizon to other HHE schemes

SASTA proves to be lethal for PASTA. However, PASTA is not the only HHE scheme over $\mathbb{F}_p$, and this gives rise to an important question: can this attack be extended to the other HHE schemes over $\mathbb{F}_p$? Can this also be extended to other $\mathbb{Z}_2$ schemes?. In this section, we address this by examining other state-of-the-art schemes over $\mathbb{F}_p$, namely MASTA [HKC+20], HERA [CHK+21], and RUBATO [HKL+22]. We also extend our analysis to RASTA [DEG+18], a scheme over $\mathbb{Z}_2$.

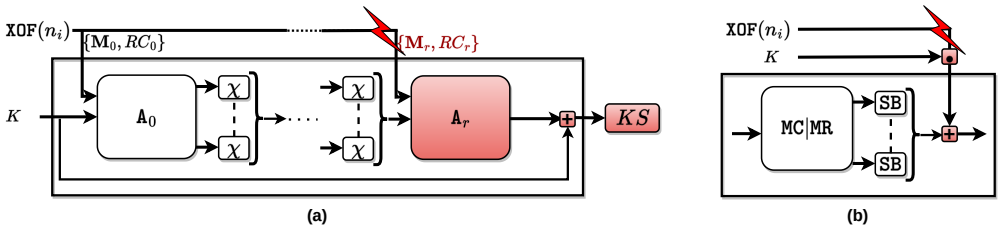

Figure 7: The fault injection points for inducing a fault in (a) Masta permutation before final A_r and (b) HERA or RUBATO permutation before final add-round-key.

5.1 Analyzing MASTA under the proposed DFA

The MASTA [HKC+20] scheme design was introduced concurrently with PASTA and can be viewed as a direct adaptation of RASTA [DEG+18], originally defined over $\mathbb{Z}_2$, to $\mathbb{F}_p$. As outlined in Subsection 2.2, MASTA incorporates two primary functions in its permutation: the χ s-box and the affine layer $\mathtt{A}_i$ (illustrated in Figure 3 (a)). The matrix used for multiplication within $\mathtt{A}_i$ is generated in a manner similar to PASTA, ensuring it is invertible by design. In the last round r, if the state with t elements after the last S-box is denoted as X_r, then the final result following the permutation is expressed as follows:

$$KS^j = \mathtt{A}_r(X_r)^j + K^j \pmod p \;\forall\; 0 \leq j < t \tag{8}$$

$$KS^j = (\mathbf{M}_r \cdot X_r)^j + RC_r^j + K^j \pmod p \tag{9}$$

Under both fault injection points (FIP case 1 and 2), the fault is induced in the matrix $\mathbf{M}_r$, as depicted in Figure 7 (a), and the resulting differential is as presented below:

$$\Delta KS^j = (\mathbf{M}_r \cdot X_r)^j - (\mathbf{M}'_r \cdot X_r)^j \pmod p \tag{10}$$

$$\Delta KS^j = (\Delta\mathbf{M}_r \cdot X_r)^j \pmod p \tag{11}$$

Theorem 3. *Given $\Delta \boldsymbol{M}_r$ is known and invertible, the* SASTA *attack strategy leads to a unique key-recovery for* MASTA.

Proof. Similar to the case of PASTA, the only unknowns in the above equations are X_r and K. Given that the difference matrix is invertible and known, Equation 11 can be used to retrieve X_r. There is no truncation done here; instead, the key is added in the end, as shown in Equation 9. This provides us with an interesting opportunity to directly retrieve the key K by using the previously obtained X_r. Hence, no round inversion is required, and because matrix $\mathbf{M}_r$ is invertible, a unique key is obtained. □

Regarding constraints on $\Delta\mathbf{M}_r$ in Theorem 3, we can apply the same argument as that used for Pasta, asserting that with a high probability (close to 1), $\Delta\mathbf{M}_r$ will be invertible. Hence, Masta is susceptible under the same attack model as Pasta, with the key recovery process significantly simplified due to the last add-key operation.

5.2 Extending attack strategy to Hera

Hera [CHK+21], as discussed in Subsection 2.2, takes a slightly different design approach and uses the key for a pseudo-key-schedule, where the `XOF` output is multiplied with the key and then added to the state. This is followed by the affine layer, which has constant matrices, unlike Pasta and Masta. A fault in the last `XOF` call for Hera (Figure 7 (b)) would result in the following equations:

$$KS^j = X_r^j + (RV_r \cdot K)^j \pmod p \tag{12}$$

$$\Delta KS^j = (RV_r' \cdot K)^j - (RV_r \cdot K)^j \pmod p \tag{13}$$

$$\Delta KS^j = (\Delta RV_r \cdot K)^j \pmod p \tag{14}$$

where RV_r represents the random vector generated from the last `XOF` call, and X_r denotes the state before this operation.

Theorem 4. *Given ΔRV_r is known, and fully diffused, the* SASTA *attack strategy leads to a unique key-recovery for* Hera.

Proof. The invertibility constraint was required in the case of Pasta and Masta because the differential was a matrix ($\Delta\mathbf{M}_r$). In the case of Hera, the known difference is simply a vector ΔRV_r, hence recovering key K, using Equation 14, only involves solving simple linear equations. Thus, the invertibility constraint is relaxed in this setting, and similar to Masta, no round inversion is required, which further simplifies the key recovery. □

Under FIP case 1, Hera is completely vulnerable as RV_r will be homomorphically encoded and multiplied with the encrypted key K (Equation 12). Consequently, the fault diffuses perfectly across the whole state, resulting in full key recovery. However, under FIP case 2, there is a fully diffused condition on RV_r because, unlike Pasta and Masta, the `XOF` output is not followed by any matrix generation. Hence, the diffusion in RV_r completely depends on the diffusion obtained from `XOF`.

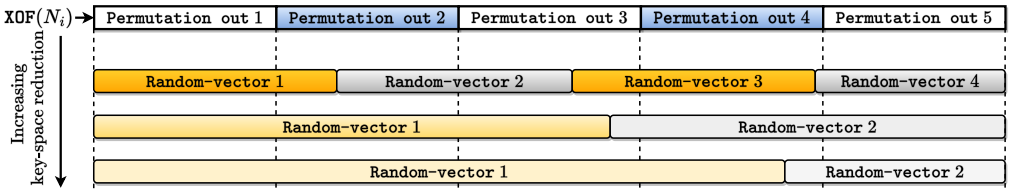

Figure 8: This figure shows that more Keccak permutations for random vector generation of Hera make it easier to target a specific vector without affecting others, leading to a higher reduction in the keyspace.

To understand why this may not always lead to full diffusion, let us look at how the `XOF` calls work. It retrieves values from the Keccak output, which are then rejection sampled. If the Keccak output state is fully consumed, then another permutation is done, and a new output is generated. Note that primes that lead to a high rate of rejection sampling would require more permutations. Similarly, large moduli would also require more permutations. Hence, the random vector obtained would be distributed across many

permutations. On the other hand, primes that have a low rejection rate will require fewer permutations. This is shown in Figure 8.

When we induce a fault, we have to ensure that we do not make the `XOF` results required in the previous rounds, faulty. Hence, we must induce the fault in the first permutation, which is exclusive to the `XOF` call for RV_r. This would diffuse the fault in all the remaining permutations as well. If the coefficients in RV_r are more distributed, then more values would be faulty. Consequently, most key values can be recovered, and the reduced keyspace becomes susceptible to brute-force attacks. To summarize, our analysis reveals that in the case of Hera, prime moduli leading to a high rate of rejection sampling are more vulnerable under SASTA than primes with a lower rate of rejection sampling. Note that this analysis is specific to fault type- Keccak instruction skip. Another fault type or location, such as matrix counter skip or skipping the matrix multiplication instruction, could also have a similar effect. This will have a success probability of 1 without any reliance on prime-size or rejection rate.

5.3 The curious case of Rubato

The design of Rubato [HKL+22] is heavily influenced by Hera and Pasta. Even though the design is very similar, Rubato cannot support schemes over integers $\mathbb{Z}_p$ like BGV and BFV that are supported by Pasta, Masta, and Hera. It can only support a scheme that does approximate arithmetic- CKKS. The reason being use of additive Gaussian noise (GSN) at the end of the HHE routine, which modifies the Equation 12 as follows:

$$KS^j = X_r^j + (RV_r \cdot K)^j + GSN \pmod{p} \tag{15}$$

The GSN introduced during encryption is not removed during decryption and remains in the data throughout the homomorphic operations. This inherent noise leads to a precision loss, rendering the data unsuitable for precise integer arithmetic and confining its utility to CKKS [CKKS17]. On the other hand, this error enhances the security guarantees, transforming the problem into an LWE problem. This is precisely why the authors opt for low multiplicative depth per round and fewer rounds in the scheme. After fault injection, the differential is expressed as follows:

$$\Delta KS^j = (\Delta RV_r \cdot K)^j + GSN \pmod{p} \tag{16}$$

Observe that this differential is an LWE sample. In the DFA fault scenario, the attacker possesses knowledge of plaintext and ciphertext pairs. However, the introduction of noise disrupts this assumption, transforming the message into an unknown value due to the addition of GSN. Hence, SASTA cannot exploit Rubato.

5.4 Applicability of SASTA to Rasta

Until now we explored state-of-the-art HHE scheme over $\mathbb{F}_p$, however we note that the SASTA technique is quite general and can also be applied to schemes over $\mathbb{F}_2$. The only dependency being, that there are much fewer invertible matrices in $\mathbb{Z}_2$ [KKW08], therefore, it is more probable that the differential might not result in unique key-recovery and could require multiple such fault attack or an exhaustive search over the reduced key-space.

We analyzed Rasta [DEG+18] to demonstrate this. As discussed in Section 2, Rasta has the same design principles as Masta [HKC+20] with slightly different `SB` and $p = 2$. The implementation of Rasta does not use Keccak, hence here we target the matrix counter increment for the single known instruction skip. This fault replays the previous round's $(r-1)$ matrix used for matrix-vector multiplication in the last round r. It also results, in the desired difference for SASTA and leads to a unique key-recovery, using the Theorem 3 for $p = 2$. In case the attacker does not retrieve an invertible matrix difference,

then in the worst case the attack needs to be mounted again. However, there is a way to get around it because the parameters for generating matrices are public. Hence, it can be known beforehand if the difference between the final and previous matrix will result in an invertible matrix or not. The attacker should use this information and mount the attack only when there is possibility of the difference matrix being invertible.

Next, we will present the fault injection attack experiment on KECCAK permutation and discuss the details of key recovery.

6 The Ambush: Fault Attack Demonstration

SASTA does not require a specific fault, and the attack strategy solely relies on the ability to determine the faulty output, for example, based on a known fault location. Hence, there are no constraints on the type of fault or the specific operation it affects. It can be introduced at any point within the extended execution of the KECCAK permutation. Moreover, we will illustrate how, with a Simple Power Analysis (SPA) or emulation, we can estimate the position of fault to a few possible locations. As a result, we relax the requirement on knowledge of the instruction affected by the fault.

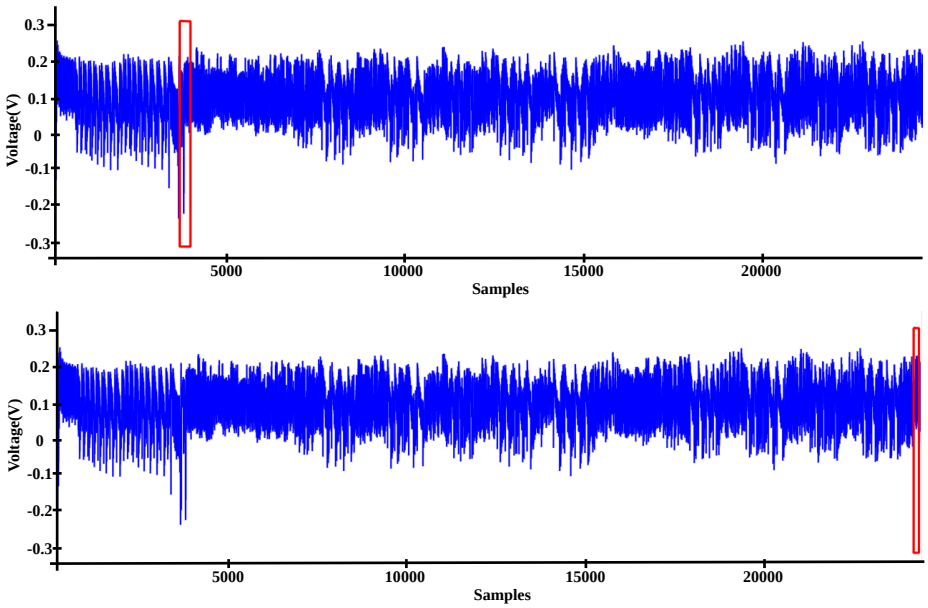

Figure 9: The sample fault injection attack positions targeted in our work. The first position is the beginning of KECCAK permutation, and the second is where the results of permutation are written to the final buffer.

To successfully demonstrate fault injection on the KECCAK permutation, we use the ChipWhisperer-Lite CW1173 evaluation board, paired with the standard FIPS202 implementation of KECCAK. Our target platform is CWLITEXMEGA, and we employ clock-glitching to induce faults. The design is running at the default clock frequency of 7.38 MHz. The fault is mounted on the client's device at FIP 2 with target set to the KECCAK based `XOF`. The KECCAK function call has three sub-calls: absorb, permute, and squeeze. The absorb call places the input into the KECCAK state, the permute call runs the permutation, and the squeeze call returns the output.

Figure 9 illustrates the two specific locations where we injected our attack. The first location corresponds to the start of the KECCAK Permute operation. Injecting a fault here resulted in the skipping of this particular instruction, causing the output from the

preceding permutation to be replayed. The second known fault injection point is towards the latter stages of the KECCAK Permute operation. This led to a shifted version of the non-faulty KECCAK output, providing a known fault. For the latter case, we made minor changes to the FIPS202 implementation to simplify the attack. Since, the input to the KECCAK is known and public, and the instruction skipped is also known, the difference Δ (in $\mathbf{M}_r$ or RV_r) required for key recovery also becomes known.

We further tested the effectiveness of basic countermeasures, such as duplication. We only demonstrate the attack on the client's device owing to higher practical feasibility. However, with the help of a malicious insider the same can be applied on the server side as well. Next, we delve into identifying successful fault injections and relaxing the need for known faults.

6.1 Identifying successful fault injection

From our experimentation, it became evident that not all fault injections yield a faulty result. Identifying this is easy as the difference will be zero. When an effective fault is induced, our model necessitates knowledge of the specific operation affected. Numerous works in the literature [DRPR19, MDP+20, BFP19, BH22] have shown that this can be done with high precision. We also demonstrated this in our experimentation above.

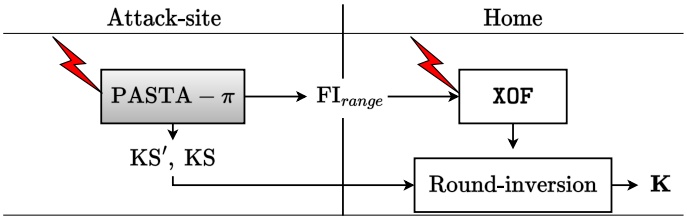

Figure 10: Correctly identifying the injected fault.

In cases where attackers lack access to precise fault injection techniques, an alternative approach can be employed, as shown in Figure 10. Attackers can induce an unknown fault and rely on power traces to estimate the fault injection range within which the fault is triggered, as depicted by the coloured boxes in Figure 9. The attacker can now leave the attack site and go home. Now, they can simulate the fault in an `XOF` emulation, recovering potential faulty outputs and corresponding key guesses. Thereafter, with just a single fault-free keystream result (KS), they can uniquely recover the actual key. This works because the attack model and setting are known to the attacker, and standard KECCAK implementation is used.

Our attack demonstration is conducted on the microcontroller. However, it is essential to note that precise faults, such as bit flips within the KECCAK state, can be mounted on FPGA implementations as well. These faults will also result in the differentials of the same form as desired by SASTA. Therefore, both software and hardware implementations require protection against SASTA.

6.2 Empirical results for key recovery

This section explores the attack specifics and outcomes. We first examine how the attacker determines the time-offset for fault injection. Subsequently, we explore the duration and range of the attack, ultimately resulting in unique key recovery.

In traditional DFA attacks, it is hard for the attacker to precisely pinpoint the time of execution of vulnerable operations and, thus, the best time to inject a fault. In the case of SASTA, we get around this because the only non-constant portion in the execution

of all the HHE schemes is the rejection sampling, which for a known prime has a fixed average rejection rate. It enables the attacker to estimate a good time offset, and we reiterate that an attacker can induce the fault anytime in the entire Keccak permutation. Thus providing a broad attack window with a 92% success probability for faults resulting in an exploitable differential. Although we focus on the Keccak permutation, skipping instructions like matrix multiplication or generation further extends the attack range.

Depending on the scheme and prime selection, the attack duration, during which faults can be induced, varies from 4% to 18% of the total execution time of HHE on the client side, offering potential extension to earlier rounds as an intriguing future scope. The high success probability of the fault, attributed to its non-specific instruction skip nature, makes it particularly effective when induced during the first Keccak permute call in the Matrix/Vector generation phase. To execute the attack, we utilize plain HHE implementations provided by the authors of Pasta for all four schemes. Furthermore, we have verified that our results remain unaffected by the end-to-end protocol, as our attack specifically targets the symmetric-key encryption and decryption part of the HHE-FHE protocol.

Following the attack overview outlined in Subsection 4.3, our primary target is the Pasta scheme, which exhibits unique key recovery. This characteristic also applies to Rasta, Masta, and Hera. As detailed in Table 1, we evaluate the 128-bit secure variants Pasta-3/4, Masta-4/5, Rasta-5/6, and Hera-5, achieving unique key recovery (100% success probability once the single known fault has been realized). It only requires the encryption of one message block on the client side. Our key recovery takes significantly less time than prior works, and our proposed attack model allows a weaker and more practical attacker setting. It is important to note that although we use the instruction skip fault to demonstrate the attack's feasibility, SASTA is not restricted to instruction skip and can use other type of faults, for example, arithmetic faults during Keccak permutation or glitches in storage of Keccak intermediate states.

In the next section, we will explore efficient countermeasures to mitigate SASTA and analyze their associated costs.

7 Countermeasure analysis

In this section, we will discuss several countermeasures that can be applied to protect against SASTA. These countermeasures are not algorithmic and come at the cost of latency, area consumption, or both. While they offer immediate solutions, a more comprehensive and algorithmic approach would be ideal for securing HHE against potential fault attacks, and we recognise this as a promising direction.

7.1 Pregeneration and Storage

One potential countermeasure against SASTA involves the offline pregeneration and storage of `XOF` output. This technique works effectively for Rasta and Hera as the matrices hold boolean data in the former case and in the latter case, the `XOF` output is always a vector and is never used for matrix generation. However, for Pasta and Masta, this is challenging due to the large amount of `XOF` output and matrix storage needed. To put this into perspective, the storage demand for just one Pasta-4 permutation matrix is approximately 22KB. Given that this data changes with each iteration, generating and storing such substantial volumes of data can lead to significant storage expenses, especially from the client's perspective. Thus, more effective mitigation techniques are essential.

Another way to achieve this is by interleaving the random vector generation for all matrices. In doing so, any fault introduced would propagate across all vectors, effectively scrambling the entire state much earlier and rendering the fault unexploitable. However, a

bit-flip fault affecting a specific random vector would lead to a fault attack, as the matrix generation would spread the fault. Hence, we conclude that this is not the most effective countermeasure for Pasta and Masta.

7.2 Redundancy and fault detection

In the long run, redundancy would be a more suitable countermeasure to make the attack setting hard for an attacker. This duplicate computation will only be required for the last round. Along with redundancy, we propose to employ multiple checks at every stage of the last round to ensure fault detection. Note that redundancy is not the ultimate solution as the attacker can bypass the checks, but they need many precise faults. This increases the required strength for mounting an effective fault attack. With a certain degree of redundancy, we can ensure the security of the design in a realistic scenario.

7.3 Infective Countermeasure

Since fault detection can be bypassed by a stronger adversary, infective countermeasure [GSSC15] proves to be more reliable. Under this countermeasure, we employ an `XOR`-based detection mechanism. The last round is duplicated, and at every stage, the two duplicate computations are `XOR`ed. This `XOR`ed result is `AND`ed with two random vectors, and the result is `XOR`ed back to the two duplicated states. This happens after both the Affine and `Mix` transform. If there is no fault, the `XOR` result will always be zero. Hence, `AND` with the random vector will also be zero, and the last `XOR` would not have any effect on the states. On the contrary, if there is a fault, then the values `XOR`ed back to the state would introduce random unexploitable garbage to the result.

7.4 Could masking be a countermeasure?

Masking is a technique employed to protect against side-channel attacks, such as differential power analysis. This involves splitting the secret into multiple shares and processing each share separately, ensuring that these shares are never combined directly at any intermediate stage. In our case, the secret is the key; hence, all the computations involving the key would have to be masked. This involves the entire key-dependent block. Since the scheme only has additions/subtractions or multiplication, additive masking will be used. In contrast, the `XOF` is not key-dependent and hence will not be masked. Therefore, SASTA would not be affected by masking.

There is a possibility that the designers intend to mask the `XOF` as well. Keccak is generally masked using boolean masking [GSM17]. Hence, a boolean to arithmetic (B2A) conversion will also be required. We would like to note that SASTA can still be applied on either of the masked Keccak permutations. This is because we do not need a specific differential; our attack works with *any* differential. Hence, this will induce the same effect required for mounting SASTA. Note that, in the current literature, no masking scheme is proposed for any HHE scheme. Hence, we only analyze the effect of our proposed masking technique, which is additive making for key-dependent operations, and conclude that masking is not an effective countermeasure against SASTA.

Countermeasure Performance Evaluation

We report performance evaluation results based on the reference implementation provided by the authors of Pasta [DGH+23]. The baseline implementation consumes 274,203 clock cycles on average for 100,000 executions of Pasta $-\pi$. The implementation results are reported for the 12th Gen Intel i7 CPU. In Figure 11, we estimate the cost of countermeasures regarding latency and required extra storage. For storage requirements,

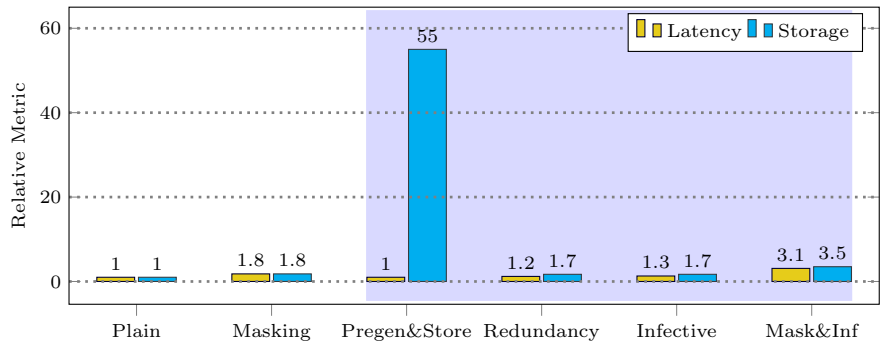

Figure 11: A comparison of latency and storage for different PASTA implementations (with or without countermeasures). The four highlighted implementations are effective countermeasures against SASTA.

we estimate the cost in terms of maximum state and intermediate variable storage required at any point in time during the PASTA-π execution.

Since there is no masking scheme for HHE, we note that the entire key-dependent operations will be duplicated for the two key shares, whereas the `XOF` operations will not be masked. Using this observation, we estimate the latency and area cost for first-order masked implementation. From Figure 11, it is evident that the Redundancy and Infective countermeasures offer the best trade-offs regarding latency increase and storage requirements for defence against SASTA. For protection against both SCA and SASTA, we propose using both masking and infective countermeasures with slightly extra overhead.

8 Discussion and Conclusion

In this work, we introduced a novel fault model for attacking HHE schemes over the prime field. We applied this model to launch a differential fault attack- SASTA on PASTA [DGH+23]. Remarkably, even in the nonce-respecting setting, a single fault is sufficient to achieve full key recovery. This poses a grave threat to privacy, as it renders all previous and future communications, with the server for homomorphic computations, vulnerable. To our knowledge, this is the first instance of exploiting a nonce-respecting fault model setting for HHE schemes. We also validate its feasibility by conducting an actual fault attack on the KECCAK permutation.

Furthermore, we demonstrated that other state-of-the-art HHE schemes, namely RASTA [DEG+18], MASTA [HKC+20] and HERA [CHK+21], are also susceptible to SASTA. We also analyzed that RUBATO [HKL+22], by relying on the LWE problem, maintains robustness against SASTA. Nevertheless, this resilience comes at the cost of high precision loss. We further discuss conventional countermeasure techniques and also analyze that masking will be ineffective.

We restrict this study to differential fault analysis. However, it would be interesting and useful to explore the vulnerability of these systems for other fault analysis models and even examine them under side-channel analysis in the future. Implementing HHE on FPGAs and analyzing the countermeasure costs also presents an interesting future direction. In summary, this work advances the field by introducing a novel attack model, demonstrating vulnerabilities in existing HHE schemes and lays the foundation of fault analysis of HHE without nonce-reuse. We hope that this will inspire the development of robust hybrid homomorphic encryption schemes in the future.

Acknowledgement

This work was supported in part by CONFIDENTIAL-6G EU project (Grant No: 101096435), DST - OEAD GRANT for Indo-Austrian research movement, and the State Government of Styria, Austria – Department Zukunftsfonds Steiermark.

References

[AKS20] Aikata, Banashri Karmakar, and Dhiman Saha. DESIV: differential fault analysis of SIV-Rijndael256 with a single fault. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2020, San Jose, CA, USA, December 7-11, 2020*, pages 241–251. IEEE, 2020.

[AMK+23] Aikata Aikata, Ahmet Can Mert, Sunmin Kwon, Maxim Deryabin, and Sujoy Sinha Roy. Reed: Chiplet-based scalable hardware accelerator for fully homomorphic encryption, 2023.

[BBH+22] Alexandros Bampoulidis, Alessandro Bruni, Lukas Helminger, Daniel Kales, Christian Rechberger, and Roman Walch. Privately connecting mobility to infectious diseases via applied cryptography. *Proc. Priv. Enhancing Technol.*, 2022(4):768–788, 2022.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.

[Beu22] Ward Beullens. Mayo: Practical post-quantum signatures from oil-and-vinegar maps. In Riham AlTawy and Andreas Hülsing, editors, *Selected Areas in Cryptography*, pages 355–376, Cham, 2022. Springer International Publishing.

[BFP19] Claudio Bozzato, Riccardo Focardi, and Francesco Palmarini. Shaping the glitch: Optimizing voltage fault injection attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):199–224, 2019.

[BGV11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. *Electron. Colloquium Comput. Complex.*, page 111, 2011.

[BH22] Jakub Breier and Xiaolu Hou. How practical are fault injection attacks, really? *IEEE Access*, 10:113122–113130, 2022.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, 2012.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.

[CCF+18] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. *J. Cryptol.*, 31(3):885–916, 2018.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

[CHK+18] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, volume 11349 of *Lecture Notes in Computer Science*, pages 347–368. Springer, 2018.

[CHK+21] Jihoon Cho, Jincheol Ha, Seongkwang Kim, ByeongHak Lee, Joohee Lee, Jooyoung Lee, Dukjae Moon, and Hyojin Yoon. Transciphering framework for approximate homomorphic encryption. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part III*, volume 13092 of *Lecture Notes in Computer Science*, pages 640–669. Springer, 2021.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437. Springer, 2017.

[Cla07] Christophe Clavier. Secret external encodings do not prevent transient fault analysis. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 181–194. Springer, 2007.

[CMdG+21] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. Labeled PSI from homomorphic encryption with reduced computation and communication. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 1135–1150. ACM, 2021.

[DEG+18] Christoph Dobraunig, Maria Eichlseder, Lorenzo Grassi, Virginie Lallemand, Gregor Leander, Eik List, Florian Mendel, and Christian Rechberger. Rasta: A cipher with low anddepth and few ands per bit. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 662–692. Springer, 2018.

[DEK+18] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. SIFA: exploiting ineffective fault inductions on symmetric cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):547–572, 2018.

[DGH+23] Christoph Dobraunig, Lorenzo Grassi, Lukas Helminger, Christian Rechberger, Markus Schofnegger, and Roman Walch. Pasta: A case for hybrid homomorphic encryption. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(3):30–73, 2023.

[DRPR19] Jean-Max Dutertre, Timothée Riom, Olivier Potin, and Jean-Baptiste Rigaud. Experimental analysis of the laser-induced instruction skip fault model. In Aslan Askarov, René Rydhof Hansen, and Willard Rafnsson, editors, *Secure IT Systems - 24th Nordic Conference, NordSec 2019, Aalborg, Denmark, November 18-20, 2019, Proceedings*, volume 11875 of *Lecture Notes in Computer Science*, pages 221–237. Springer, 2019.

[DS01] K. Davidson and S. J. Szarek. Local operator theory, random matrices and banach spaces. In *Handbook of the geometry of Banach spaces, Vol. I,*, pages 317–366, 2001.

[FJLT13] Thomas Fuhr, Éliane Jaulmes, Victor Lomné, and Adrian Thillard. Fault attacks on AES with faulty ciphertexts only. In Wieland Fischer and Jörn-Marc Schmidt, editors, *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013*, pages 108–118. IEEE Computer Society, 2013.

[FSK+21] Axel Feldmann, Nikola Samardzic, Aleksandar Krastev, Srini Devadas, Ron Dreslinski, Karim Eldefrawy, Nicholas Genise, Christopher Peikert, and Daniel Sanchez. F1: A fast and programmable accelerator for fully homomorphic encryption (extended version), 2021.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 144, 2012.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme.* PhD thesis, Stanford University, USA, 2009.

[GPP11] Jian Guo, Thomas Peyrin, and Axel Poschmann. The PHOTON family of lightweight hash functions. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 222–239. Springer, 2011.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.

[GSM17] Hannes Groß, David Schaffenrath, and Stefan Mangard. Higher-order side-channel protected implementations of KECCAK. In Hana Kubátová, Martin Novotný, and Amund Skavhaug, editors, *Euromicro Conference on Digital System Design, DSD 2017, Vienna, Austria, August 30 - Sept. 1, 2017*, pages 205–212. IEEE Computer Society, 2017.

[GSSC15] Shamit Ghosh, Dhiman Saha, Abhrajit Sengupta, and Dipanwita Roy Chowdhury. Preventing fault attacks using fault randomization with a case study on AES. In Ernest Foo and Douglas Stebila, editors, *Information Security and Privacy - 20th Australasian Conference, ACISP 2015, Brisbane, QLD, Australia, June 29 - July 1, 2015, Proceedings*, volume 9144 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2015.

[GVBP+22] Robin Geelen, Michiel Van Beirendonck, Hilder V. L. Pereira, Brian Huffman, Tynan McAuley, Ben Selfridge, Daniel Wagner, Georgios Dimou, Ingrid Verbauwhede, Frederik Vercauteren, and David W. Archer. BASALISC: Flexible Asynchronous Hardware Accelerator for Fully Homomorphic Encryption, 2022.

[HKC+20] Jincheol Ha, Seongkwang Kim, Wonseok Choi, Jooyoung Lee, Dukjae Moon, Hyojin Yoon, and Jihoon Cho. Masta: An he-friendly cipher using modular arithmetic. *IEEE Access*, 8:194741–194751, 2020.

[HKL+22] Jincheol Ha, Seongkwang Kim, ByeongHak Lee, Jooyoung Lee, and Mincheol Son. Rubato: Noisy ciphers for approximate homomorphic encryption. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I*, volume 13275 of *Lecture Notes in Computer Science*, pages 581–610. Springer, 2022.

[HS18] Shai Halevi and Victor Shoup. Faster homomorphic linear transformations in helib. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 93–120. Springer, 2018.

[JLK+21] Wonkyung Jung, Eojin Lee, Sangpyo Kim, Jongmin Kim, Namhoon Kim, Keewoo Lee, Chohong Min, Jung Hee Cheon, and Jung Ho Ahn. Accelerating fully homomorphic encryption through architecture-centric analysis and optimization. *IEEE Access*, 9:98772–98789, 2021.

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha P. Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 1651–1669. USENIX Association, 2018.

[KGSR23] Anup Kumar Kundu, Shibam Ghosh, Dhiman Saha, and Mostafizar Rahman. Divide and rule: Difa - division property based fault attacks on PRESENT and GIFT. In Mehdi Tibouchi and Xiaofeng Wang, editors, *Applied Cryptography and Network Security - 21st International Conference, ACNS 2023, Kyoto, Japan, June 19-22, 2023, Proceedings, Part I*, volume 13905 of *Lecture Notes in Computer Science*, pages 89–116. Springer, 2023.

[KKK+22] Sangpyo Kim, Jongmin Kim, Michael Jaemin Kim, Wonkyung Jung, John Kim, Minsoo Rhu, and Jung Ho Ahn. BTS: An Accelerator for Bootstrappable Fully Homomorphic Encryption. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, page 711–725, New York, NY, USA, 2022. Association for Computing Machinery.

[KKW08] Caroline Kim, John Y. Kim, and Dr. Guofang Wei. Invertibility probability of binary matrices team. 2008.

[LN16] Patrick Longa and Michael Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In Sara Foresti and Giuseppe Persiano, editors, *Cryptology and Network Security*, pages 124–139, Cham, 2016. Springer International Publishing.

[LYK21] Duc-Phong Le, Sze Ling Yeo, and Khoongming Khoo. Algebraic differential fault analysis on SIMON block cipher. *IACR Cryptol. ePrint Arch.*, page 436, 2021.

[MAK+23] Ahmet Can Mert, Aikata, Sunmin Kwon, Youngsam Shin, Donghoon Yoo, Yongwoo Lee, and Sujoy Sinha Roy. Medha: Microcoded Hardware Accelerator for computing on Encrypted data. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(1):463–500, 2023.

[MCJS19] Pierrick Méaux, Claude Carlet, Anthony Journault, and François-Xavier Standaert. Improved filter permutators for efficient FHE: better instances and implementations. In Feng Hao, Sushmita Ruj, and Sourav Sen Gupta, editors, *Progress in Cryptology - INDOCRYPT 2019 - 20th International Conference on Cryptology in India, Hyderabad, India, December 15-18, 2019, Proceedings*, volume 11898 of *Lecture Notes in Computer Science*, pages 68–91. Springer, 2019.

[MDP+20] Alexandre Menu, Jean-Max Dutertre, Olivier Potin, Jean-Baptiste Rigaud, and Jean-Luc Danger. Experimental analysis of the electromagnetic instruction skip fault model. In *15th Design & Technology of Integrated Systems in Nanoscale Era, DTIS 2020, Marrakech, Morocco, April 1-3, 2020*, pages 1–7. IEEE, 2020.

[Mic10] Daniele Micciancio. A first glimpse of cryptography's holy grail. *Commun. ACM*, 53(3):96, 2010.

[MJSC16] Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. Towards stream ciphers for efficient FHE with low-noise ciphertexts. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 311–343. Springer, 2016.

[PA11] G. Bertoni, J. Daemen, M. Peeters and G. Van Assche. The Keccak reference. Round 3 submission to NIST SHA-3, 2011.

[RBM21] Dibyendu Roy, Bhagwan N. Bathe, and Subhamoy Maitra. Differential fault attack on kreyvium & FLIP. *IEEE Trans. Computers*, 70(12):2161–2167, 2021.

[RKMR23] R. Radheshwar, Meenakshi Kansal, Pierrick Méaux, and Dibyendu Roy. Differential Fault Attack on Rasta and FiLIP_{DSM}. *IEEE Trans. Computers*, 72(8):2418–2425, 2023.

[Rud06] Mark Rudelson. Norm of the inverse of a random matrix. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 487–496. IEEE Computer Society, 2006.

[SC16] Dhiman Saha and Dipanwita Roy Chowdhury. Encounter: On breaking the nonce barrier in differential fault analysis with a case-study on PAEQ. In *CHES*, volume 9813 of *Lecture Notes in Computer Science*, pages 581–601. Springer, 2016.

[Spr] Daan Sprenkels. The Kyber/Dilithium NTT. https://dsprenkels.com/ntt.html.

[SSL15] Kazuo Sakiyama, Yu Sasaki, and Yang Li. *Security of Block Ciphers - From Algorithm Design to Hardware Implementation*. Wiley, 2015.

[TMA11] Michael Tunstall, Debdeep Mukhopadhyay, and Subidh Ali. Differential fault analysis of the advanced encryption standard using a single fault. In Claudio A. Ardagna and Jianying Zhou, editors, *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication - 5th IFIP WG 11.2 International Workshop, WISTP 2011, Heraklion, Crete, Greece, June 1-3, 2011. Proceedings*, volume 6633 of *Lecture Notes in Computer Science*, pages 224–233. Springer, 2011.

[VZB+22] Navid Vafaei, Sara Zarei, Nasour Bagheri, Maria Eichlseder, Robert Primas, and Hadi Soleimany. Statistical effective fault attacks: The other side of the coin. *IEEE Trans. Inf. Forensics Secur.*, 17:1855–1867, 2022.

[ZFL+22] Fan Zhang, Tianxiang Feng, Zhiqi Li, Kui Ren, and Xinjie Zhao. Free fault leakages for deep exploitation: Algebraic persistent fault analysis on lightweight block ciphers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(2):289–311, 2022.