SASTA: Ambushing Hybrid Homomorphic Encryption with a Single Fault

Aikata Aikata[1], Ahaan Dabholkar[2], Dhiman Saha[3], Sujoy Sinha Roy[1]
[1]*IAIK, Graz University of Technology,* [2]*Purdue University,* [3]*Indian Institute of Technology Bhilai*

{*aikata,sujoy.sinharoy*}*@iaik.tugraz.at, adabholk@purdue.edu, 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 has been hindered by substantial computational and communication overheads, particularly on the client's side. The Hybrid Homomorphic Encryption (HHE) protocol was developed to mitigate these issues. However, the susceptibility of HHE to strong attacks, specifically physical attacks, has been largely unexplored. While physical attacks like the Differential Fault Analysis (DFA) have proved very effective in the field of symmetric cryptography, prior works have largely relied on strong assumptions like nonce reuse, limiting their feasibility in a real-world setting.**

In this work, we introduce a novel attack- SASTA, which presents, to the best of our knowledge, the first generalized analysis of HHE under DFA. Our analysis uncovers a significant limitation of the HHE protocol where a single fault leads to complete key recovery not only for the standard scheme-AES but also for the new HHE tailored Symmetric Encryption (SE) schemes – RASTA, PASTA, MASTA, and HERA. We further extend SASTA to effectively target Authenticated Transciphering protocols. Unlike prior works, the key advantage of SASTA is that it does not require nonce reuse.

We demonstrate a proof-of-concept of our attack on an off-the-shelf ATXmega128D4-AU microcontroller running HHE firmware and mount end-to-end key recovery attacks. Finally, we discuss conventional countermeasures to defend against SASTA. Our work highlights that despite HHE's advantages of improving performance and reducing communication overhead, further analysis of its security guarantees is required.

***Index Terms*—Homomorphic Encryption, Hybrid Encryption, Transciphering, Fault attacks,** AES-GCM, PASTA, HERA, RASTA, RUBATO

1. Introduction

The world is witnessing a concerning surge in daily data breaches, typically directed at large data storage centres and servers. Such breaches allow malicious parties access to vast amounts of private information. The key reason being, even though data is encrypted for communication, the servers inevitably have to decrypt it for computation or storage. Not only does this allow the server to see the client's data, but also attackers in the event of a data breach. To address this growing security and privacy concern, there is a pressing need for privacy-preserving storage and computation.

Homomorphic Encryption (FHE) schemes [10], [11], [15], [16], [23], [28] offer such privacy preserving capabilities. However, their widespread adoption is inhibited by significant computational and communication overheads. This is because FHE encryption schemes transform data into substantially larger polynomials and rely on costly polynomial arithmetic for performing computations on the encrypted data. While server-side computations can be expedited through dedicated hardware accelerators [2], [24], [27], [41], [49], the same approach is infeasible for client-side applications due to their high cost. Moreover, the transmission of these large polynomials incurs huge communication overheads.

Hybrid Homomorphic Encryption (HHE), also known as *transciphering* [52], was introduced to tackle these problems. The main idea behind HHE is to replace the expensive homomorphic data encryption methods with symmetric key encryption. In the HHE setup, clients encrypt their data with a symmetric key before sending it to the server. The server then homomorphically evaluates the decryption circuit on the symmetric ciphertext to transform it into a homomorphic ciphertext (asymmetric). The obtained FHE ciphertext can then be used to perform the required computations on the server. As a result, this method saves significant communication and computation requirements on the client.

Over the years, researchers have realized that standard symmetric key schemes (e.g., AES) operating on boolean data have a huge performance overhead. As a result new HHE tailored Symmetric Encryption (SE) schemes – PASTA [21], MASTA [34], HERA [17], and RUBATO [35], have been proposed. Unlike standard schemes, these operate over integers in prime fields ($\mathbb{F}_p$) and offer better performance for real-world applications like machine learning.

Recent works [1], [6] have proposed *authenticated* transciphering (AT) to further enhance the capabilities of HHE by ensuring the integrity and authenticity of encrypted data. They achieve this by using symmetric schemes (e.g., AES-GCM) in Authenticated Encryption with Associated Data (AEAD) mode.

Despite its promise, the HHE protocol itself and the

new SE schemes have not undergone extensive cryptanalysis, especially in the context of *physical attacks* such as Differential Fault Analysis (DFA). Physical attacks on cryptographic implementations are attacks that exploit the physical characteristics of the device running the primitive to recover secret parameters involved in its computation. These attacks can involve measuring the execution time and power consumption or actively disturbing a cryptographic computation to recover sensitive data. These attacks are often more efficient than classical cryptanalytic attacks and are considered a serious threat by device vendors.

In this work, we present SASTA, a novel DFA attack on HHE to recover private cryptographic keys used by the client device. We use a *single* pair of fault-free and faulty ciphertexts to construct differentials that subsequently help retrieve private key information. Our key observation is a significant vulnerability in the HHE protocol, which makes it susceptible to a DFA attack even in a *nonce-respecting* setting. We also extend the SASTA-based fault analysis to the AT setting (utilizing AES-GCM) and present an end-to-end key-recovery.

The SASTA attack proceeds in three main stages: First, the attacker induces a single fault during the SE encryption routine on the client device, resulting in the generation of erroneous ciphertexts, which are sent to the server. Second, the attacker constructs a differential using the results returned by the server. Since the server operates on a faulty ciphertext, the resulting computation shows a disparity in the output, which is decrypted on the client's side. Finally, with this differential, the attacker mounts a key-recovery attack, breaking user privacy.

To date, there have been few DFA studies [54], [55] on HHE tailored SE schemes [13], [19], [46], [47]. These works are limited by their sole focus on the symmetric facet of the HHE protocol. Moreover, they rely on strong assumptions like nonce reuse, which are prohibited in their respective specifications [17], [19], [21], [34], precluding classical differential analysis. On the other hand, SASTA takes into account the homomorphic aspects of the HHE protocol, which allows it to bypass the nonce–reuse assumption, making it a realistic attack while adhering to the design specifications. Finally, while prior works target schemes operating on boolean data [13], [19], [46], [47], we analyze SASTA for all recent schemes – PASTA [21], MASTA [34], HERA [17], and RUBATO [35] that operate in prime fields and are consequently more efficient. An analysis of these schemes under DFA is unexplored yet much needed due to the vast potential for deployment of privacy-preserving applications. Our results are summarized in Table 1.

Field	Target Scheme	Sec.	nonce resp.	#F	Time	Attack
$\mathbb{Z}_2$	Flip_{530}	80	No	1	39hr	[55]
	Kreyvium	128		3	5min	[55]
	Filip_{430}	80		1	751hr	[54]
	RASTA-6	80		1	96hr	[54]
	RASTA-5	128	Yes	1	0.1s	**Sec.4.5**
	RASTA-6	128		1	0.12s	**Sec.4.5**
	AES-GCM	128		1	5s	**Sec.4.6**
	AES-GCM	256		3	45min†	**Sec.4.6**
$\mathbb{Z}_p$	PASTA-3	128		1	0.5s	**Sec.4.1**
	PASTA-4	128		1	0.21s	**Sec.4.1**
	MASTA-4	128		1	0.09s	**Sec.4.2**
	MASTA-5	128		1	0.07s	**Sec.4.2**
	HERA-5	128		1	0.005s	**Sec.4.3**

TABLE 1. COMPARISON OF SASTA WITH PRIOR WORKS [54], [55] ANALYZING HHE TAILORED SE UNDER DFA. THE AVERAGE TIME IS REPORTED FOR KEY-RECOVERY ON A 2GHZ CLOCK AND 4GB RAM PROCESSOR. #F REFERS TO THE NUMBER OF FAULTS REQUIRED AND † STANDS FOR RESULTS REPORTED IN [4].

Contributions

In this work, we investigate both types of HHE enabling SE schemes: (1) the standard scheme AES-GCM, and (2) the new high-performance scheme proposals- RASTA, PASTA, MASTA, HERA, and RUBATO. Analysing the latter is more challenging due to a lack of prior DFA literature on SE schemes defined over $\mathbb{F}_p$. We list our main contributions below:

- SASTA- **First protocol level nonce respecting attack on** HHE: To the best of our knowledge, we are the first to show the viability of fault attacks on HHE while conforming to scheme specifications. Our attack technique operates under a *nonce-respecting* setting, thus requiring a significantly weaker attacker than prior DFA works.
- **First DFA of SE schemes over $\mathbb{F}_p$:** We develop the first key-recovery attack utilizing SASTA for breaking the new HHE tailored SE schemes- PASTA [21], MASTA [34], HERA [17]. This also extends to RASTA [19] defined over $\mathbb{Z}_2$.
- **A new attack on Authenticated Transciphering (AT):** We extend our attack and, for the first time, show how AT [1] (based on AES-GCM) becomes vulnerable to DFA by leveraging SASTA.
- **Attack Proof-of-Concept:** We demonstrate our attack using an 8-bit off-the-shelf micro-controller ATXmega128D4-AU set as target using the ChipWhispererLite CW1173 board. This attack requires just a *single known fault*, resulting in complete key-recovery. Additionally, we have integrated the firmware for the new HHE tailored SE scheme- HERA to validate SASTA, further amplifying the scope of our investigation.
- **Countermeasure Analysis:** We finally provide a comprehensive analysis of potential strategies and techniques to strengthen the security of HHE protocol against SASTA.

2. Background and Related Work

Notation: $\mathbb{Z}_2$ refers to Boolean values and $\mathbb{Z}_p$ refers to integers modulo $p > 2$. $\mathbb{F}_p$ is used to denote prime fields for a prime modulus p. We will denote numbers (integers and

real) with small letters (e.g., x), vectors of numbers with capital letters (e.g., X) except message m and ciphertext c, and matrices with bold and capital letters (e.g., $\mathbf{M}$). Subscripts X_i are used for naming different matrices or vectors, and superscripts X^i are used for indexing coefficients in a vector or rows in a matrix. Throughout the paper, variables with an apostrophe sign (e.g., $c', C', \mathbf{M}'$) refer to faulty values resulting from fault injection. Finally, the subscripts FHE or SE imply the value is encrypted using FHE or SE. The red coloured values- (e.g., x) imply faulty computation or result.

2.1. Homomorphic Encryption

Before we delve into the specifics of 'Hybrid' Homomorphic Encryption (also known as transciphering), we provide a brief introduction to 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 [28], and since then, several FHE schemes have been introduced by researchers, such as BGV [11], BFV [10], [23], CKKS [14], [15], and TFHE [16]. 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 ' [50].

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 $100,000\times$ [39]. Transmitting these large ciphertexts and performing operations on them is quite expensive. To address this challenge, researchers have developed specialized hardware accelerators [2], [24], [27], [41], [49], 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.

2.2. Hybrid Homomorphic Encryption (HHE)

HHE addresses the increased communication and computational complexity incurred because of FHE-encryption by essentially using a symmetric key-based encryption method that can be decrypted in a homomorphic manner. The top-level overview of the HHE protocol is depicted in Figure 1, and a stepwise description is illustrated in Table 2. It proceeds as follows.

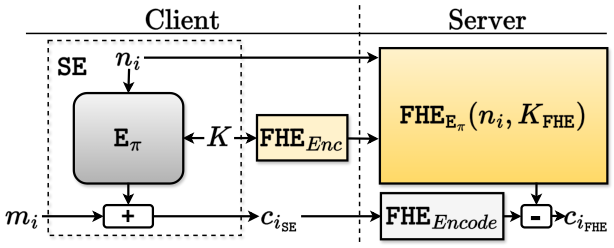

Figure 1. Workflow of HHE is illustrated here. The homomorphically encrypted key (K_{FHE}) is communicated only once. After this, multiple small ciphertexts ($c_{i_{\mathrm{SE}}}$) encrypted using SE are sent along with nonce. The server performs a 'homomorphic SE decryption circuit evaluation' (HSD)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

Client	Server
Initialization	
Generate keys K and $\{sK, pK\}$ $K_{FHE} = \mathrm{FHE}_{Enc}(K)_{pK}$ Send pK and K_{FHE} $\rightarrow$	$\rightarrow$ Save keys pK and K_{FHE}
Encryption using SE	
$c_{i_{SE}} = m_i + \mathrm{E}_\pi(K, n_i)$ Send $c_{i_{SE}}$, n_i, and $f()$ $\rightarrow$	$\rightarrow$ $c_{i_{SE}}$, n_i, and $f()$
Homomorphic SE *Decryption Circuit Evaluation (*HSD*)*	
	$c_{i_{FHE}} = c_{i_{SE}} - \mathrm{FHE}_{E_\pi}(n_i, K_{FHE})$
Function Evaluation	
	Compute $c'_{i_{FHE}} = f(c_{i_{FHE}})$
$c'_{i_{FHE}}$ $\leftarrow$	$\leftarrow$ Send $c'_{i_{FHE}}$
Result Decryption	
$f(m_i) = \mathrm{FHE}_{Dec}(c'_{i_{FHE}})_{sK}$	

TABLE 2. A SIMPLIFIED ALGORITHMIC DESCRIPTION OF THE HHE PROTOCOL. HERE, i REFERS TO i-TH ITERATION OF THE HHE PROTOCOL.

- The client has two types of keys, the symmetric encryption key K and asymmetric private and public keys $\{sK, pK\}$ for FHE. These keys are long-term keys and remain constant for a long time. The client needs to protect both K and sK because knowledge of 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.
- Next, the client uses the public key pK to encrypt K using $\mathrm{FHE}_{Encrypt}$, and sends this to the server at the beginning itself[1]. After this, whenever clients need to store or process data on the cloud, they use SE Encryption to encrypt the message blocks m_i using K and send the resultant ciphertext $c_{i_{\mathrm{SE}}} = m_i + \mathrm{E}_\pi$ to the server along with the public nonce n_i. E_π refers to the key-dependent permutation that results in a keystream, which is added to plaintext for encryption.
- The server uses the encrypted key $\mathrm{K}_{\mathrm{FHE}}$ along with the nonce n_i to evaluate the SE decryption circuit homomorphically ($\mathrm{FHE}_{\mathrm{E}_\pi}$). This results in a ciphertext ($c_{i_{\mathrm{FHE}}}$), which is only homomorphically

1. Note that the client also computes evaluation keys and sends them to the server in the initialization phase. We do not mention them as they are irrelevant to our investigation.

encrypted under key sK. This can now be stored on the server or processed ($f()$) as the client desires.
- The client can retrieve the original/resultant ciphertext and use sK to decrypt the result.

SE schemes that can only operate on Boolean data (e.g., AES, RASTA [19]) entail a substantial performance decline on the server side. Hence, for better performance and application support, researchers have developed schemes like MASTA [34], PASTA [21], HERA [17], and RUBATO [35] that can directly operate over prime fields $\mathbb{F}_p$, enabling operations on real or integer plaintexts, and offering cost-effective communication and efficient (client-side) encryption. Thus, in our work, we focus on both these scheme types to demonstrate the broad applicability of our attack. Section A contains the design overview of these new SE schemes.

2.3. Authenticated Transciphering (AT)

In the homomorphic setting, AT [1], [6] is used to ensure data integrity; that is, the data sent by the client is the same as that received by the server for computation. We stress on the fact that conventional FHE schemes do not offer data integrity and so the use of AT offers an additional benefit over other HHE related advantages.

In a symmetric setting, the client sends encrypted data along with a tag to the server. The server generates a tag from the received data and matches it with the received tag to verify data integrity. However, in the homomorphic setting, the server can only generate an *encrypted* tag using encrypted key (K_{FHE}). As a result, it cannot do tag matching to verify integrity. Consequently, the server has to send the encrypted tag back to the client who has to verify whether the two Tags match. This is briefly described in Table 3.

First, AT was analyzed by the authors in [6]. They utilized Grain128-AEAD [37], and this was converted to a TFHE [16] ciphertext on the server side. This is followed by a recent work [1], where the authors report benchmarks for authenticated transciphering using standard symmetric encryption schemes such as AES-GCM and ASCON for CKKS [15] FHE scheme. The results of this work show that due to the possibility of parallel processing, AES-GCM is much faster than ASCON, which can only process plaintexts sequentially. However, even at its best AES-GCM consumes a total depth of 123, requiring many bootstrapping operations just for HSD. On the other hand, the new HHE tailored SE scheme proposals [17], [19], [21], [34], [35] do not require any bootstrapping for one encryption as they consume very less multiplicative depth, hence delivering very good performance.

In summary, the purpose of AT is to generate an additional authentication tag alongside encryption and decryption processes. This tag is intended to ensure data integrity and authenticity. The primary objective is to ensure that the authentication tag seamlessly transitions from the SE ciphertext to the FHE ciphertext. If this is not achieved with a 100% success probability, unintended decryption failures will occur.

Client	Server
Initialization	
Generate K and $\{sK, pK\}$ $K_{FHE} = \mathrm{FHE}_{Enc}(K)_{pK}$ Send pK and K_{FHE} $\rightarrow$	$\rightarrow$ Save keys pK and K_{FHE}
Encryption using SE	
$c_{i_{SE}} = m_i + \mathrm{E}_\pi(K, n_i)\ \forall 0 \leq i < l$ $\mathrm{T}_i = GCM(\mathrm{E}_\pi, c_{i_{SE}}, K)$ Send $c_{i_{SE}}$, n_i, and $f()$ $\rightarrow$	$\rightarrow$ $c_{i_{SE}}$, n_i, and $f()$
Homomorphic SE *Decryption Circuit Evaluation (*HSD*)*	
	$c_{i_{FHE}} = c_{i_{SE}} - \mathrm{FHE}_{E_\pi}(n_i, K_{FHE})$ $\mathrm{T}_{i_{FHE}} = GCM(\mathrm{FHE}_{E_\pi}, c_{i_{SE}}, K_{FHE})$
Function Evaluation	
	Compute $c'_{i_{FHE}} = f(c_{i_{FHE}})$
$c'_{i_{FHE}}, \mathrm{T}_{i_{FHE}}$ $\leftarrow$	$\leftarrow$ Send $c'_{i_{FHE}}, \mathrm{T}_{i_{FHE}}$
Result Decryption	
If $\mathrm{T}_i == \mathrm{FHE}_{Dec}(\mathrm{T}_{i_{FHE}})_{sK}$: $f(m_i) = \mathrm{FHE}_{Dec}(c'_{i_{FHE}})_{sK}$	

TABLE 3. A SIMPLIFIED ALGORITHMIC DESCRIPTION OF THE AUTHENTICATED TRANSCIPHERING PROTOCOL [1], [6]. HERE E_π REFERS TO ANY SE PERMUTATION (E.G., AES), AND i REFERS TO i-TH ITERATION OF THE HHE PROTOCOL.

2.4. Differential Fault Analysis of SE

Differential Fault Analysis is a physical attack that exploits information leaked from faulty computations on a victim's device. Faults are induced by forcing the device to operate in unexpected environmental conditions (such as high voltage surges, clock or EM glitches) and the observed differences between fault-free and faulty outputs are analyzed to reveal information about the internal states.

The standard DFA threat model consists of a victim device running encryption/decryption protocols with a secret key and an adversary with the ability to induce faults in its computations. It also allows the adversary to function in a chosen plaintext or ciphertext setting. These works [54], [55] further make an even stronger assumption by presuming the repetition of nonces to mount effective attacks. As pointed out by the authors in [20], repeating nonces always incurs a certain loss of security, and the inability to repeat the nonce renders conventional DFA techniques infeasible, as stated in [57].

The susceptibility of classical SE schemes to fault attacks has been extensively analyzed. Several key-recovery attacks have been demonstrated for RSA-CRT Signatures [8], AES [3], [5], [30], [38], [51], [61]. Prior works [3], [58], [61] show how even a single fault during AES encryption can be utilized to significantly reduce the key space, which can be made unique using known plaintext ciphertext pairs.

3. The SASTA Fault Attack

In this section, we outline the SASTA attack strategy. It is important to note that FHE and, by extension, HHE schemes are, at best, chosen-plaintext (IND-CPA) secure. In other words, a malicious (CCA2) server can simply return adaptively chosen ciphertexts, which can lead to trivial key-

recovery attacks [44]. Thus, in any HHE related setting, the server is assumed to be honest-but-curious[2].

3.1. Threat Model

We mount our attack in a known-plaintext setting, which is weaker than CPA and closely follows [43]. We adopt a standard DFA threat model from prior works [7], [42], [51], [55], [62]. The setting is that of a client communicating with an honest-but-curious server in the presence of an attacker who tries to leak the client's secrets. The attacker has the ability to observe the queries sent by the client device and the responses sent by the server during a limited time window, during which she is also able to induce a *known* glitch or a fault in the computations done on the client's device[3]. A known fault is usually induced by manipulating the operating conditions of the hardware (HW) involved in the computations [7], [42], [59], [62] and results in a skipped instruction or a flipped bit that is known to the attacker.

To utilize this model in the HHE setting (described in Section 2.2), we assume that the server computes a function f on the received (encrypted) data and responds with the (encrypted) output. This function is predefined by the client before the start of the HHE protocol, as stated in [29]. While previous works on DFA enhance the attacker's capabilities by assuming conditions like *nonce reuse*, we relax this assumption to present a more realistic adversary. Moreover, we restrict the attacker to observe only a single query-response pair to mount a successful attack. Both these requirements make our threat model significantly weaker than the ones used by prior works.

3.2. Attack Methodology

The attack proceeds in three broad phases

❶ **Faulting client's SE Encryption**: A single fault is induced by the attacker during the execution of SE encryption routine on the client (Figure 2). This fault results in the generation of erroneous ciphertexts, which are sent to the server. Since not every fault can be exploited, the attacker identifies certain Fault Injection Points (FIPs) depending on the SE primitive. These FIPs are targeted, and the resulting faults are subsequently leveraged for key-recovery.

❷ **Generating Differentials**: The attacker observes a *differential* obtained in the output returned by the server. First, the client sends a faulty message to the server, which evaluates the homomorphic decryption circuit (HSD) on it. The server then sends the function output corresponding to the message back to the client, and this output is different from the output corresponding to a fault-free message, thus forming a differential (Table 4 ⓑ). This process is further elaborated in Section 3.3.

2. Note that this assumption is not too strong as there are techniques [25], [63], [64] such as Zero knowledge proofs to ensure that the ciphertext returned by the server is not malicious and is actually the result of the requested computation for a function know to or predefined by the client.

3. Evil maid attack

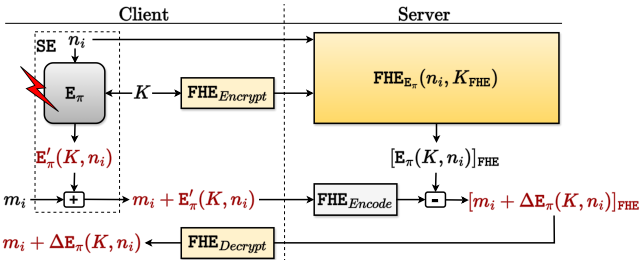

Figure 2. Utilizing SASTA for differential fault analysis of HHE tailored SE schemes or AES counter mode.

❸ **Key-Recovery**: The attacker uses the knowledge of the FIP and the generated differentials to extract the private key K that is stored on the client device and used for the HHE encryption and decryption. The private key can then be used to compromise the privacy of users on the attacked device. We detail the key-recovery methods in Section 4.

In the case of AT, the fault is induced during the very first permutation ($\mathtt{E}_\pi$) evaluation (Figure 3), which is needed only for Tag computation. Thus, the differential is obtained using the valid tag received from the server side. This differential is subsequently employed for the purpose of key-recovery.

3.3. Attacking HHE

In HHE, the client device simply performs SE (Table 2) encryption to protect the message sent to the server. The encryption involves computing key-dependent permutation ($\mathtt{E}_\pi$) and then adding it to the message (as shown in Figure 1). For schemes defined over $\mathbb{Z}_2$, instead of modular addition/subtraction, XOR is used.

The attacker induces a fault in the SE routine (illustrated in Figure 2) and observes the outputs sent by the server to generate a differential, which can subsequently be used for key-recovery, as detailed in Table 4. In the following lemmas and theorem, i refers to the i-th iteration of the HHE protocol and j is used to index message blocks (for schemes over $\mathbb{F}_p$) or bits (e.g., in AES). For simplicity, we have removed the subscripts *SE* and *FHE*.

Lemma 1 (Expression of Faulty Ciphertext). *If a fault occurs during the client-side encryption, the resulting ciphertext c'_i with t coefficients after encryption can be expressed as follows:*

$$c'^{j}_i = m^j_i + E'_\pi(K, n_i)^j \pmod p \quad \forall\ 0 \le j < t$$

Proof. In HHE, the ciphertext c^j_i is a function of the original message m^j_i and the permutation $\mathtt{E}_\pi(K, n_i)^j$ 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 as shown in Table 4 ⓐ. □

The faulty ciphertext c'^{j}_i is then sent to the server. The server performs HSD, which results in a faulty message m'_i.

Client	Server
Initialization	
Encryption using SE	
$c'_{i_{SE}} = m_i + \mathrm{E}'_\pi(K, n_i)$ ⓐ Send $c'_{i_{SE}}$, n_i, and $f()$ $\rightarrow$	$\rightarrow$ $c'_{i_{SE}}$, n_i, and $f()$
Homomorphic SE *Decryption Circuit Evaluation (*HSD*)*	
	$c'_{i_{FHE}} = c'_{i_{SE}} - \mathrm{FHE}_{E_\pi}(n_i, K_{FHE})$ ⓑ $\simeq m_{i_{FHE}} + \Delta\mathrm{E}_\pi$, where $\Delta\mathrm{E}_\pi \simeq \mathrm{E}'_\pi(K, n_i) - \mathrm{E}_\pi(K, n_i)$
Function Evaluation	
Result Decryption	
$f(m'_i) = \mathrm{FHE}_{Dec}(c'_{i_{FHE}})_{sK}$ $= f(\Delta\mathrm{E}_\pi + m_i)$ ⓒ	

TABLE 4. ALGORITHMIC DESCRIPTION OF THE FAULTY COMPUTATION DURING THE HHE PROTOCOL. FOR BREVITY, WE OMIT THE INITIALIZATION AND FUNCTION EVALUATION PARTS OF THE PROTOCOL.

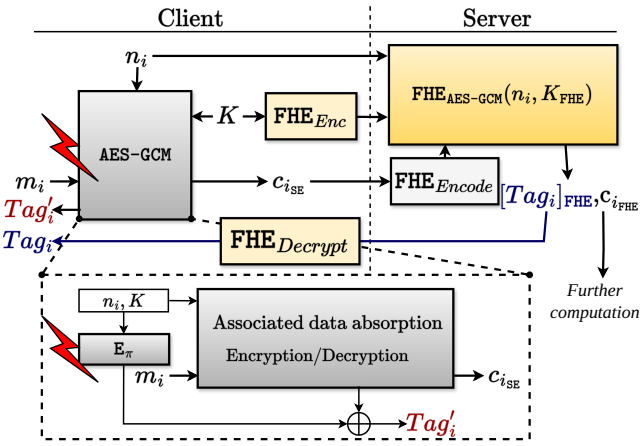

Figure 3. Exploiting SASTA for Authenticated Transciphering using AES-GCM.

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

$$m'^j = m^j + E'^j_\pi(K, n_i) - E^j_\pi(K, n_i) \pmod p \; \forall\, 0 \le j < t$$

Proof. The server, upon receiving the ciphertext c_i, performs HSD, which is essentially the subtraction of the encrypted permutation $\mathrm{E}_\pi(K, n_i)^j$ from c_i:

$$m^j_i = c^j_i - \mathrm{E}_\pi(K, n_i)^j \pmod p \; \forall\, 0 \le j < t$$
$$m'^j_i = c'^j_i - \mathrm{E}_\pi(K, n_i)^j \pmod p \; \forall\, 0 \le j < t$$

When a fault occurs during the client-side permutation, it results in a deviation from the actual computation. The server-side HSD uses the faulty ciphertext and the fault-free permutation $\mathrm{E}_\pi(K, n_i)^j$. This results in the computed message being faulty m'_i, differing from the original message m_i due to the fault-induced difference in the permutations. This is shown in Table 4 ⓑ. □

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

$$\Delta E_\pi = E'_\pi(K, n_i) - E_\pi(K, n_i) \pmod p$$

Proof. Using Lemma 1, we get a faulty ciphertext c'_i when a fault is injected in the permutation during the encryption. Lemma 2 shows that the server-side HSD of c'_i yields a homomorphically encrypted but faulty message m'_i distinct from the original message m_i. Since the original message is known to the attacker, the differential ($\Delta\mathrm{E}_\pi$) can be obtained (as shown in Table 4 ⓒ) by simply subtracting the known message m_i from m'_i.

$$m'_i - m_i = \mathrm{E}'_\pi(K, n_i) - \mathrm{E}_\pi(K, n_i) = \Delta\mathrm{E}_\pi \pmod p$$

□

Thus, we obtain a difference $\Delta\mathrm{E}_\pi$, which can be utilized for key-recovery, as detailed in Section 4. We note that generating the differential relies on the ability to compute $m'_i - m_i$ from the returned output. Hence, the function to be evaluated has to be invertible. This is further discussed in Section 7. However, this limitation does not apply to the extended attack on AT, which is independent of the choice of the function, as detailed in Section 3.4.

3.4. Extending to AT

In this section, we extend SASTA to an AT setting. As described in Section 2.3, AT is used to ensure data integrity in HHE. This is important to guarantee that the result obtained by the client is for the intended data and not faulty data. In [1], [6], the authors propose AT using schemes on boolean data (e.g., GRAIN128, AES); however, these can also be replaced by the new schemes over integers for efficient computation. Therefore, we keep the discussion in this section generic to any scheme utilizing the GCM-like mode for authenticated encryption and data integrity.

We propose mounting SASTA during the very first permutation call (E_π) of any new encryption since it is only used for tag generation and does not affect the subsequent ciphertext generation, as shown in Figure 3. The stepwise description is provided in Table 5. This is useful as a lack of dependence on ciphertexts frees us from the limitation on $f()$ being invertible (Section 3.3). Note that the associated data absorption and encryption part is shown only as a gray box in the figure as we do not alter or influence its output via our attack, and the security of AES-GCM does not rely on it being unknown. Hence, the output (ADE_i a.k.a. GHASH result) of this computation stays the same on both the client and server-side computations, and the attacker only needs to know this output.

As mentioned in the previous section, for the scheme over boolean data ($\mathbb{Z}_2$), the modular addition/subtraction shown in the equations is replaced with XOR. Following Lemma 1, the faulty tag computation on the client side can be expressed as follows (Table 5 ⓘ).

Client	Server
Initialization	
Encryption using SE	
$c_{i_{SE}} = m_i + \mathtt{E}_\pi(K, n_i)$ $ADE_i = GCM_{partial}(\mathtt{E}_\pi, c_{i_{SE}}, K)$ //Black-box $\mathtt{Tag}'_i = \mathtt{E}'_\pi(K, n_i) \oplus ADE_i$ ⓘ Send $c_{i_{SE}}$, n_i, and $f()$ $\rightarrow$	$\rightarrow c_{i_{SE}}$, n_i, and $f()$
Homomorphic SE *Decryption Circuit Evaluation (*HSD*)*	
Function Evaluation	
Result Decryption	
If $\mathtt{Tag}'_i == \mathtt{FHE}_{Dec}(\mathtt{Tag}_{i_{FHE}})_{sK}$: ⓘⓘ $f(m_i) = \mathtt{FHE}_{Dec}(c'_{i_{FHE}})_{sK}$	

TABLE 5. ALGORITHMIC DESCRIPTION OF THE FAULTY COMPUTATION DURING AUTHENTICATED TRANSCIPHERING.

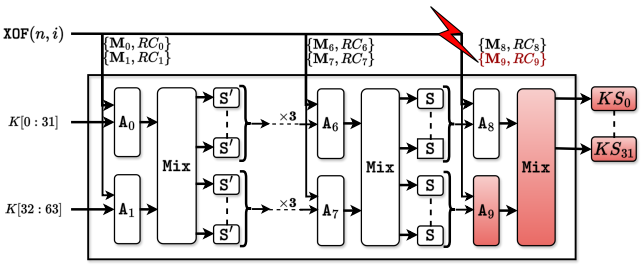

Figure 4. The attack strategy and fault injection point for PASTA-4 permutation ($KS = \mathtt{E}_\pi(K, n_i)$).

$$\mathtt{Tag}'_i = ADE_i + \mathtt{E}'_\pi(K, n_i) \pmod p$$

This is followed by the encrypted (fault-free) tag computation on the server side, as shown below:

$$\mathtt{Tag}_i = ADE_i + \mathtt{E}_\pi(K, n_i) \pmod p$$

Since the tag is encrypted, the server sends it to the client for verification, and the client obtains a differential without requiring nonce-repetition thus keeping with our threat model (shown in Table 5 ⓘⓘ).

$$\begin{aligned}\Delta\mathtt{Tag}_i &= \mathtt{Tag}'_i - \mathtt{Tag}_i \pmod p \\ &= \mathtt{E}'_\pi(K, n_i) - \mathtt{E}_\pi(K, n_i) = \Delta\mathtt{E}_\pi \pmod p\end{aligned}$$

This obtained differential can subsequently be used for key-recovery. Observe that the differential obtained here is the same as Theorem 1. Therefore, the same key-recovery techniques can be utilized.

4. Leveraging SASTA: Key-Recovery Case Studies

In this section, we present an end to end key-recovery attack on a variety of ciphers with unique constructions to show the impact of SASTA. We provide detailed case studies for each of the targeted ciphers- AES, PASTA, MASTA, HERA, and RASTA. In each study, we identify critical Fault Injection Points (FIPs) which can help exploit the differential (Section 3.3 or Section 3.4) for key-recovery.

4.1. Cracking PASTA's Defenses

The PASTA-4 permutation is shown in Figure 4. It consists of four rounds each comprising an Affine Layer $\mathtt{A}()$, a Mixing layer $\mathtt{Mix}()$ and S-box $\mathtt{S}'/\mathtt{S}()$. The state is treated as two concatenated vectors $(X_i||X_j)$ and is initialized with the secret key. Affine transform performs matrix-vector (state) multiplication and round constant addition to the state. The Mix is used to combine the results of the two partial vectors in the state. All of these transforms are completely reversible and can be traced back to the initial state (secret key). Hence, to prevent trivial round-inversion, at the very end a truncation is done, and only one of the vectors in the state is given as the final output ($KS = \mathtt{E}_\pi(K, n_i)$). For a brief understanding of PASTA permutation please refer to Section A.

Several candidate FIPs exist for inducing a fault such that it diffuses widely throughout the PASTA-4 permutation (Figure 9). For instance, introducing a fault before the final $\mathtt{S}$ operation will result in its diffusion across the entire state due to the subsequent matrix multiplications ($\mathtt{A}_{8/9}$) and $\mathtt{Mix}$ steps. However, post-experimentation, we concluded that this differential alone will not lead to a unique key-recovery, as multiple keys can fulfill the condition of obtaining a specific differential after a fault injection. Thus, it would necessitate multiple such faults to eliminate potential key guesses. Therefore, we resort to seeking alternate fault injection points.

We induce a fault in the matrix to be multiplied during step $\mathtt{A}_9$, as depicted in Figure 4. If we denote the input state after the last $\mathtt{S}$ operation as $X_8||X_9$, the fault-free result is as follows.

$$\mathtt{E}_\pi(K, n_i) = \mathtt{Trunc}(\mathtt{Mix}(\mathtt{A}_8(X_8), \mathtt{A}_9(X_9))) \quad (1)$$

$$\mathtt{E}_\pi(K, n_i) = 2 \cdot (\mathbf{M}_8 \cdot X_8 + RC_8) + (\mathbf{M}_9 \cdot X_9 + RC_9) \quad (2)$$

The $\mathtt{Trunc}$ operation prevents easy round inversion for key-recovery. After fault injection, we obtain the following:

$$\mathtt{E}'_\pi(K, n_i) = 2 \cdot (\mathbf{M}_8 \cdot X_8 + RC_8) + (\mathbf{M}'_9 \cdot X_9 + RC_9) \quad (3)$$

$$\Delta\mathtt{E}_\pi = \Delta\mathbf{M}_9 \cdot X_9 \quad (4)$$

$$X_9 = \Delta\mathbf{M}_9^{-1} \cdot \Delta\mathtt{E}_\pi \quad (5)$$

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

Proof. The permutation consists of matrix multiplications with invertible matrices, and the remaining linear operations are inherently invertible. Therefore, a one-to-one mapping exists 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 $\mathtt{XOF}$ 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. □

Theorem 2 (**Unique Key-recovery**). *Given that* $\Delta \boldsymbol{M}_9$ *is known and invertible, 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 $\mathtt{E}'_\pi(K, n_i)$, $\Delta\mathtt{E}_\pi$, and consequently $\mathtt{E}_\pi(K, n_i)$ due to faulty client-side computation of permutation and the differential obtained from the server ($\Delta\mathtt{E}_\pi$). The matrices $\mathbf{M}_8, \mathbf{M}_9$ and round constants RC_8, RC_9 are generated using the Extendable Output Function (XOF), e.g., KECCAK. The inputs to XOF are nonce and counter, which are public. Hence, these variables are also known.

If we can use a fault to recover intermediate states X_8, X_9, it would lead to a complete key-recovery (Lemma 3). Truncation (Equation 2) prevents this inversion from the final state to the initial state (K) by hiding half of the final state. To do this, we will recover X_9 using Equation 5, given that $\Delta\mathbf{M}_9$ is known and invertible. We can use this information to recover X_8 from Equation 3, mitigating the truncation, thus leading to a unique key-recovery. □

$\Delta\mathbf{M}_9$ is the difference caused by fault injection in the matrix $\mathbf{M}_9$, and its value is known due to known-fault position (the location of the instruction-skipped or bit-flip). For ensuring invertibility with high probability [40], we discuss next how we choose appropriate fault injection points.

Locating the Interesting FIPs

Ensuring $\Delta\mathbf{M}_9$ is invertible after a fault is crucial for determining the appropriate FIP. For this, we look for FIPs that lead to a full matrix diffusion as evidenced by studies [18], [40], [56] which indicate the probability of random n dimensional square matrices being invertible is close to one ($\prod_{k=1}^{n}(1-p^{k-1-n}) < 1-\frac{1}{p}$ where p is the modulus). Since the fault induced is fairly random, with high probability, we can ensure $\Delta\mathbf{M}_9$ has no linear dependencies and is thus invertible.

To induce such a fault, we identify the KECCAK (discussed in item C) *permutation* (XOF) as a viable option. A fault in the XOF function, just before it generates $\mathbf{M}_9$, leads to a full matrix diffusion. This exploits KECCAK's diffusion property against HHE. Observe that this does not require a specific fault, for example, skipping a very particular instruction. The sole requirement is knowing which fault is induced so the attacker can know the value of $\Delta\mathbf{M}_9$.

4.2. Analyzing MASTA

The MASTA [34] scheme design was introduced concurrently with PASTA and can be viewed as a direct adaptation of RASTA [19], originally defined over $\mathbb{Z}_2$, to $\mathbb{F}_p$. As outlined in Section A, MASTA incorporates two primary functions in its permutation: the χ S-box and the affine layer $\mathtt{A}_i$ (illustrated in Figure 10). 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

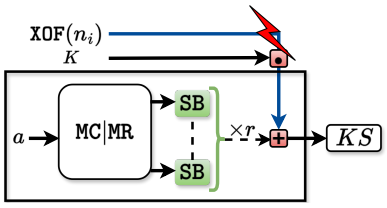

Figure 5. The fault injection points for inducing a fault in HERA or RUBATO permutation before final add-round-key.

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:

$$\mathtt{E}_\pi(K, n_i) = \mathtt{A}_r(X_r) + K \pmod p \tag{6}$$
$$\mathtt{E}_\pi(K, n_i) = \mathbf{M}_r \cdot X_r + RC_r + K \pmod p \tag{7}$$

The fault is induced in the matrix $\mathbf{M}_r$ (Figure 10), and the resulting differential is as presented below:

$$\Delta\mathtt{E}_\pi = \mathbf{M}_r \cdot X_r - \mathbf{M}'_r \cdot X_r \pmod p \tag{8}$$
$$\Delta\mathtt{E}_\pi = \Delta\mathbf{M}_r \cdot X_r \pmod p \tag{9}$$

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 9 can be used to retrieve X_r. No truncation is done here; instead, the key is added in the end, as shown in Equation 7. This gives us an interesting opportunity to directly retrieve the key K 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 is used for PASTA, asserting that with a high probability (close to 1), $\Delta\mathbf{M}_r$ will be invertible. Hence, MASTA is susceptible to SASTA, with the key-recovery process significantly simplified due lack of truncation and the last add-key operation.

4.3. Extending Attack Strategy to HERA

HERA [17], as elaborated in Section B, takes a slightly different design approach and uses a pseudo-key-schedule, where the XOF output vector (RV) is multiplied with the key and then added to the state (X). This is followed by the affine layer, which has constant matrices, unlike PASTA and MASTA. A fault in the last XOF call utilizing RV_r for state X_r (Figure 5) would result in the following equations:

$$\mathtt{E}_\pi(K, n_i) = X_r + RV_r \cdot K \pmod p \tag{10}$$
$$\Delta\mathtt{E}_\pi = RV'_r \cdot K - RV_r \cdot K \pmod p \tag{11}$$
$$\Delta\mathtt{E}_\pi = \Delta RV_r \cdot K \pmod p \tag{12}$$

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 12, 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. □

In previous cases, a small fault in the XOF output vector led to a fully diffused matrix due to the matrix generation. This implies that diffusion is dependent on both XOF and matrix generation. However, in the case of HERA, the XOF output vector (RV) is directly utilized instead of being employed to generate a matrix. Consequently, diffusion is solely dependent on the XOF. Hence, we rely on attacks that skip the first few values produced by the XOF call to populate the vector. This results in a shifted vector that offers adequate diffusion. It can be noted that this FIP is a small subset of possible FIPs for PASTA and MASTA. Equation C explains how the big set of FIPs used in previous cases can also be utilized for HERA.

4.4. The Curious Case of RUBATO

The design of RUBATO [35] (Section C) 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 10 as follows:

$$\mathtt{E}_\pi(K, n_i) = X_r + RV_r \cdot K + GSN \pmod p \tag{13}$$

After fault injection, the differential is expressed as follows:

$$\Delta \mathtt{E}_\pi = \Delta RV_r \cdot K + GSN \pmod p \tag{14}$$

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 [15]. 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.

Observe that the differential obtained 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 for plain modes. However, since no noise can be added to tag computation during the homomorphic authenticated decryption process, RUBATO can be exploited if GCM-like authenticated encryption modes are used.

4.5. 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 $\mathbb{Z}_2$. The only dependency being that there are much fewer invertible matrices in $\mathbb{Z}_2$ [40]. Therefore, it is more probable that the differential might not result in unique key-recovery, which would require an exhaustive search over the reduced key space. Alternatively, multiple faulty ciphertexts can also help converge to a unique key.

We analyzed RASTA [19] to obtain the differential as follows.

$$\mathtt{E}_\pi(K, n_i) = \mathtt{A}_r(X_r) \oplus K \tag{15}$$
$$\mathtt{E}_\pi(K, n_i) = \mathbf{M}_r \cdot X_r \oplus RC_r \oplus K \tag{16}$$
$$\Delta \mathtt{E}_\pi = \mathbf{M}_r \cdot X_r \oplus \mathbf{M}'_r \cdot X_r \tag{17}$$
$$\Delta \mathtt{E}_\pi = \Delta \mathbf{M}_r \cdot X_r \tag{18}$$

As discussed in Section A, RASTA has the similar design principles as MASTA [34]. The implementation of RASTA also uses KECCAK for XOF, hence making it the appropriate FIP as done in the previous cases. It results in the desired difference for SASTA and leads to a unique key-recovery, using the Theorem 3.

4.6. Exploiting AES via Known Approaches

With AES having been a standard for over two decades, numerous studies have analyzed it using DFA [3]–[5], [30], [38], [51], [58], [61]. These works also include both single or multiple-fault-based key-recovery techniques [3], [61], [30], [38], [51]. Most of these techniques do not even rely on known fault conditions. The FIPs employed in these studies can be utilized to realize SASTA. Consequently, SASTA can be employed to extend the same attacks in an HHE setting with weaker assumptions, due to non-reliance on nonce-reuse.

It is crucial to mention that DFA methods requiring multiple fault injections also depend on both fault-free and faulty ciphertext pairs for the same nonce. However, with SASTA, the nonce-reuse assumption is relaxed, making such attacks feasible in practical scenarios. Therefore, in the context of AES, SASTA does not restrict the number of possible fault injections to one, and existing unique key-recovery techniques that require multiple fault injections for AES-256 [4], [5], [30], [38], [51] can be utilized. This is because these works only require faulty and fault-free ciphertext pairs under the same plaintext and nonce, as provided by SASTA.

Overall, in this section, we elaborated how SASTA can be utilized to mount DFA resulting in full key-recovery on

the standard (AES-GCM) as well as state-of-the-art constructions (RASTA, PASTA, MASTA, and HERA). For all the above case studies, we only considered fault attack surfaces on client-side. However, for the sake of completeness, we also analyzed possible FIP on the server-side computations detailed in Figure C. We emphasize that attacking the client's device is a more realistic setting and is generally the target of all prior works.

Next, we will present the fault injection attack experiment resulting in practical key-recovery.

5. The Ambush: Fault Attack Demonstration

HHE employs symmetric key schemes for reduced computation and communication overhead, enabling it to be executed on resource-constrained embedded devices. We implement SASTA on one such constrained embedded device, specifically the ATXmega128D4-AU on the ChipWhisperer-Lite CW1173 evaluation board[4]. It is an 8-bit Harvard architecture RISC single-chip off-the-shelf microcontroller, which runs lightweight schemes like the TINYAES. Similar to prior fault injection works [59], we also utilize the opensource ChipWhisperer toolchain[5].

Given the limitations of this device, it is infeasible to perform an entire conventional client-side FHE computation. However, the use of HHE makes this possible[6]. To illustrate the potential of HHE in enabling FHE utilizable encryption on resource-constrained devices, we have successfully implemented and executed the HERA algorithm on the ATXmega128D4-AU. This is essential for the development of secure and efficient encryption solutions for a wide range of applications. Next, we will discuss how we induce known faults in KECCAK, and ambush the implementation of HERA.

5.1. Fault Injection Technique

The SASTA attack model does not require a specific fault, and the attack strategy solely relies on the ability to determine the faulty output. Several works in the literature [9], [12], [22], [48], [59] have validated its feasibility. 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 can estimate the position of the fault within a few possible locations (discussed in Section 5.2). As a result, we can further relax the requirement for knowledge of the instruction affected by the fault.

To successfully demonstrate fault injection on the KECCAK permutation we use the FIPS202 standard implementation [26]. Our target platform is CWLITEXMEGA, and we employ clock-glitching to induce faults. The design runs as firmware at the default clock frequency of 7.38 MHz. The

4. https://rtfm.newae.com/Targets/CW303%20XMEGA/

5. https://github.com/newaetech/chipwhisperer

6. The ChipWhisperer opensource toolchain runs AES firmware and hence can support AES-GCM mode.

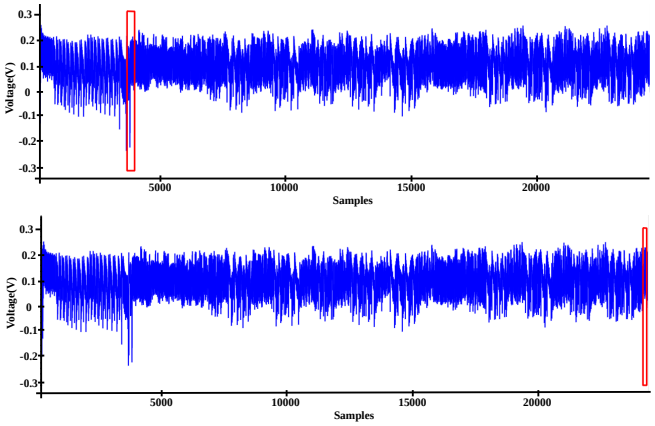

Figure 6. The sample fault injection attack positions targeted in our work. The first position is the beginning of KECCAK permutation, and the second is where the results of permutation are written to the final buffer

fault is mounted on the client's device with the 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 6 illustrates the power trace and 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 skipping this particular instruction, causing the output from the preceding permutation to be refed. 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, after the fault injection, the difference, Δ (in $\mathbf{M}_r$ or RV_r) required for key-recovery also becomes known.

These two known fault injection points serve to demonstrate the feasibility of known fault injections on microcontrollers. Similarly, voltage glitches [59] can also be utilized to mount precise attacks during storage of KECCAK intermediate states. It is essential to note that precise faults, such as bit flips within the KECCAK state, can also be mounted on FPGA implementations, showcasing versatility. We further tested the effectiveness of duplication countermeasures.

5.2. Attack and Key-recovery

To showcase a fault injection resulting in full key-recovery, we analyze the HERA firmware. This is because the available interesting FIPs for HERA are a subset of those required for RASTA, PASTA, and MASTA (as discussed in Section 4.3). Hence, an attack on HERA would also showcase practicality of SASTA on the other schemes. Figure 7 displays the power trace for the final Add-round-key operation. As we aim for the entire state to be faulty, we introduce a known fault at the beginning of the computation.

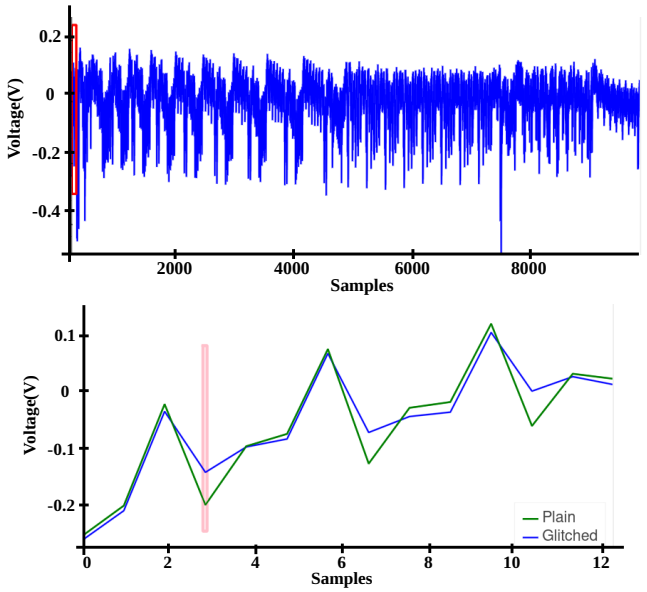

Figure 7. The first diagram illustrates the fault induction during the last HERA ARK operation. The second diagram presents an enlarged view of the normal and faulty computations following the clock glitch. Please note that these traces have been extracted from the original extensive plot, and the x-axis is relative.

This known fault skips the initial few KECCAK squeeze outputs, causing the resulting XOF output to shift and provide the necessary fault diffusion. We then employ the strategy discussed in Section 4.3 and utilize this known single fault to recover the secret key. This attack does not require any nonce repetition due to the HHE setting, making it highly effective.

Our attack does not necessitate power traces shown in Figure 7. These power traces are merely used to demonstrate the fault injection points, and its effect on the power consumption. The power analysis attacks are more complex and require specialized equipment and expertise than fault attacks. Therefore, as described in the attack model, our attack only requires an attacker to be capable of inducing faults during the execution of SE Encryption.

5.2.1. 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 (ΔE_π) will be zero. When an effective fault is induced, our model necessitates knowledge of the specific operation affected. Numerous works in the literature [9], [12], [22], [48], [59] have shown that this can be done with high precision. We also demonstrated this in our experimentation above.

In cases where attackers lack access to precise fault injection techniques, an alternative approach can be employed. Attackers can induce an unknown fault and rely on the fault injection range within which the fault is triggered. Since the firmware is known, the attacker can simulate all the faults in an XOF emulation within the attack range, recovering potential faulty outputs and corresponding key guesses. Finally, they can uniquely recover the actual key with just a single fault-free encryption result.

In conclusion, while the successful execution of the HERA algorithm on the ATXmega128D4-AU highlights the potential of HHE, the SASTA ambush emphasizes the importance of reevaluating these solutions, particularly the HHE protocol, against realistic attack settings. This is crucial to ensure their robustness and effectiveness in real-world scenarios.

5.3. 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.

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.

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.

Following the attack overview outlined in Section 4, our primary target is the HERA scheme, which exhibits unique key-recovery. This characteristic also applies to RASTA, MASTA, and PASTA. 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. 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 protocol.

6. Countermeasure analysis

In this section, we will discuss several countermeasures that can be applied to protect HHE against SASTA. While fault attack defences for AES have already been explored, to defend the new SE constructions against the attack, the following countermeasures could be utilized. The techniques

discussed next offer immediate solutions; however, a more comprehensive and algorithmic approach would be ideal for securing HHE against potential fault attacks.

6.1. 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 the schemes itself. 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 still 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.

6.2. Redundancy and fault detection

In the long run, redundant computations would be a more suitable countermeasure to make the attack 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.

6.3. Infective Countermeasure

Since fault detection can be bypassed by a stronger adversary, infective countermeasure [31] proves to be more reliable. Towards this, we employ an XOR-based detection mechanism. The last round is duplicated, and at every stage, the two duplicate computations are XORed. This XORed result is ANDed with two random vectors, and the result is XORed back to the two duplicated states. This happens after 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 affect the states. On the contrary, if there is a fault, the values XORed back to the state would introduce random unexploitable garbage in the result.

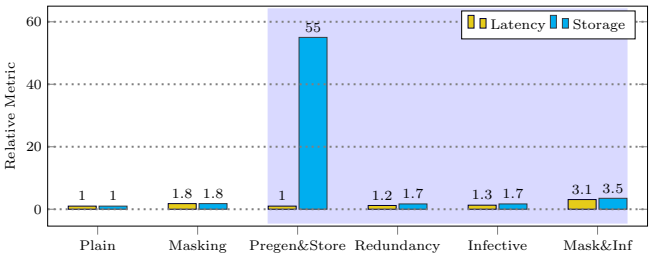

Figure 8. A comparison of latency and storage for different PASTA implementations (with or without countermeasures). The four highlighted implementations are effective countermeasures against SASTA.

6.4. How do masked implementations behave under SASTA?

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 (K); hence, all the computations involving the key would have to be masked. This involves only the key-dependent block. Since the scheme only has additions/subtractions or multiplication, additive masking will be used. In contrast, the XOF, which is the attack target, is not key-dependent and hence will not be masked. Therefore, SASTA would not be affected by masking. Note that, in the current literature, no masking scheme is proposed for any HHE scheme. Hence, we only analyze the effect of a possible additive masking.

Countermeasure Performance Evaluation

We report performance evaluation results based on the reference implementation provided by the authors of PASTA [21]. The baseline implementation consumes 274,203 clock cycles on average for 100,000 executions of PASTA-E_π. The implementation results are reported for the 12th Gen Intel i7 CPU. In Figure 8, we estimate the cost of countermeasures regarding latency and required extra storage. For storage requirements, we estimate the cost in terms of maximum state and intermediate variable storage required at any point in time during the PASTA-E_π execution. From Figure 8, it is evident that the Redundancy and Infective countermeasures offer the best latency vs memory trade-offs. For protection against both SCA and SASTA, we propose using both masking and infective countermeasures with slightly extra overhead.

6.5. Algorithmic Countermeasures for AT

To protect against an attack on AT, it is best to ensure that the tag sent by the server cannot be utilized to form the differential required for SASTA (Section 3.4).

One performance-hungry way to ensure this is by sending the tag to the server, and the server can run an encrypted

equality check computation as discussed in [1]. Hence, this incurs a computation overhead on the server side. While the attacker is able to see the result, it is not exploitable (we need the faulty tag to obtain the differential), thus safeguarding against SASTA.

However, this approach has the caveat when utilized by schemes like BGV [11], FV [23], or CKKS [15]. Since these schemes cannot compute comparisons or perform other non-polynomial operations, they use function approximations, which do not always return a fully accurate result and sometimes fail for values distributed over a large range. Consequently, there is a scope for error even when there is no fault.

Another problem associated with this approach is its reliance on the server doing an actual expensive tag verification instead of simply returning an encryption of 1's (equality check result). However, since in our threat model, we assume an honest but curious server, this is not an issue.

Another better approach that forces the server to at least correctly decrypt the data would be to pass the generated tag through an arithmetic hash function. So, on the client side, the hash of the tag is matched instead of the actual tag. Since hash functions are one-way, a faulty hash cannot be traced back to the actual fault or fault-free value. Therefore, we conclude that this approach is perhaps the best to protect against SASTA on the GCM mode of operation.

7. Limitations and Future Scope

In this section, we present the limitations of SASTA on the plain HHE protocol, which offers directions for future research. The first limitation is that SASTA cannot be directly extended to RUBATO due to its use of Gaussian error, leading to noisy but secure encryption (discussed in Section 4.4). It would be interesting to explore if this security measure can be bypassed, possibly with multiple fault injections [59].

Another limitation of our approach is its dependence on the function $f()$ being evaluated. SASTA depends on the ability to infer the message m' from the ciphertext $f(m')$ returned by the server as we need to be able to calculate $m' - m$ using the output from the server. The function $f()$ does not have to be an identity function and can be any invertible function.

However, if $f()$ is complex and reduces the output space to a few classes (e.g., in machine learning for classification), then the differential obtained cannot be exploited with a single query, and it is challenging to retrieve the input from the result of such functions. In essence, SASTA's applicability is tied to the properties of the function $f()$, which determines the ability to retrieve the input from the output.

Such a limitation also exists for prior works targeting FHE schemes [43][7]. Hence, a natural extension to our work could be generating differentials for other function classes, which warrants future investigation. However, we would like to note that for the extension on AT, we no longer depend on $f()$. The reason for this is that the tag is not part of the function evaluation but is only used to verify data integrity.

In summary, the identified limitations in SASTA open up promising avenues for future security analysis of privacy-enhancing techniques, specifically HHE protocol and AT.

7. It led to CKKS [15] being IND-CPA^D secure and not IND-CPA.

8. Conclusion

In this work we introduce a novel fault attack technique called SASTA, designed for mounting DFA-based fault injection attacks on HHE. Traditional DFA attacks require at least two ciphertext computations with fixed nonce and plaintext assumptions. This is so that fault-free and faulty ciphertext can be obtained under the same plaintext and public parameters to form an exploitable differential. However, SASTA eliminates the need for such assumptions, thus making the attack model realistic and practical.

We apply SASTA on standard scheme AES, as well as new high-performance HHE, tailored SE schemes- RASTA, PASTA, MASTA, and HERA. SASTA is also easily extendable to AT. We validate this approach experimentally by executing a fault injection attack on an off-the-shelf microcontroller running HHE firmware. This proof-of-concept demonstrates that an entire key can be recovered with just a single fault in a few seconds. Finally, we conclude by analyzing potential countermeasures to defend HHE as well as AT implementations against SASTA. We also discuss the limitations of our work, which present interesting avenues for future research.

Our work advances the field by introducing a novel attack model, demonstrating vulnerabilities in the HHE protocol, and lays the foundation of fault analysis of HHE without nonce-reuse.

Acknowledgement

This work was supported in part by State Government of Styria, Austria – Department Zukunftsfonds Steiermark and 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. The work was done during the visit of Dhiman Saha to TU Graz which was supported by DST, Government of India under the sanctioned Indo-Austria project DST/IC/Austria/P- 1/2021 (G) - (International Cooperation Division - WTZ)

References

[1] E. Aharoni, N. Drucker, G. Ezov, E. Kushnir, H. Shaul, and O. Soceanu, "Poster: Efficient AES-GCM decryption under homomorphic encryption," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, W. Meng, C. D. Jensen, C. Cremers, and E. Kirda, Eds. ACM, 2023, pp. 3567–3569. [Online]. Available: https://doi.org/10.1145/3576915.3624377

[2] A. Aikata, A. C. Mert, S. Kwon, M. Deryabin, and S. S. Roy, "Reed: Chiplet-based scalable hardware accelerator for fully homomorphic encryption," 2023.

[3] S. Ali and D. Mukhopadhyay, "A differential fault analysis on AES key schedule using single fault," in *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011, Tokyo, Japan, September 29, 2011*, L. Breveglieri, S. Guilley, I. Koren, D. Naccache, and J. Takahashi, Eds. IEEE Computer Society, 2011, pp. 35–42. [Online]. Available: https://doi.org/10.1109/FDTC.2011.10

[4] ——, "An improved differential fault analysis on AES-256," in *Progress in Cryptology - AFRICACRYPT 2011 - 4th International Conference on Cryptology in Africa, Dakar, Senegal, July 5-7, 2011. Proceedings*, ser. Lecture Notes in Computer Science, A. Nitaj and D. Pointcheval, Eds., vol. 6737. Springer, 2011, pp. 332–347. [Online]. Available: https://doi.org/10.1007/978-3-642-21969-6_21

[5] A. Barenghi, G. M. Bertoni, L. Breveglieri, M. Pellicioli, and G. Pelosi, "Fault attack on AES with single-bit induced faults," in *Sixth International Conference on Information Assurance and Security, IAS 2010, Atlanta, GA, USA, August 23-25, 2010*. IEEE, 2010, pp. 167–172. [Online]. Available: https://doi.org/10.1109/ISIAS.2010.5604061

[6] A. Bendoukha, A. Boudguiga, and R. Sirdey, "Revisiting stream-cipher-based homomorphic transciphering in the TFHE era," in *Foundations and Practice of Security - 14th International Symposium, FPS 2021, Paris, France, December 7-10, 2021, Revised Selected Papers*, ser. Lecture Notes in Computer Science, E. Aïmeur, M. Laurent, R. Yaich, B. Dupont, and J. García-Alfaro, Eds., vol. 13291. Springer, 2021, pp. 19–33. [Online]. Available: https://doi.org/10.1007/978-3-031-08147-7_2

[7] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, ser. Lecture Notes in Computer Science, B. S. K. Jr., Ed., vol. 1294. Springer, 1997, pp. 513–525. [Online]. Available: https://doi.org/10.1007/BFb0052259

[8] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults (extended abstract)," in *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, ser. Lecture Notes in Computer Science, W. Fumy, Ed., vol. 1233. Springer, 1997, pp. 37–51. [Online]. Available: https://doi.org/10.1007/3-540-69053-0_4

[9] C. Bozzato, R. Focardi, and F. Palmarini, "Shaping the glitch: Optimizing voltage fault injection attacks," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2019, no. 2, pp. 199–224, 2019. [Online]. Available: https://doi.org/10.13154/tches.v2019.i2.199-224

[10] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," in *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, ser. Lecture Notes in Computer Science, R. Safavi-Naini and R. Canetti, Eds., vol. 7417. Springer, 2012, pp. 868–886. [Online]. Available: https://doi.org/10.1007/978-3-642-32009-5_50

[11] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Fully homomorphic encryption without bootstrapping," *Electron. Colloquium Comput. Complex.*, p. 111, 2011. [Online]. Available: https://eccc.weizmann.ac.il/report/2011/111

[12] J. Breier and X. Hou, "How practical are fault injection attacks, really?" *IEEE Access*, vol. 10, pp. 113 122–113 130, 2022. [Online]. Available: https://doi.org/10.1109/ACCESS.2022.3217212

[13] A. Canteaut, S. Carpov, C. Fontaine, T. Lepoint, M. Naya-Plasencia, P. Paillier, and R. Sirdey, "Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression," *J. Cryptol.*, vol. 31, no. 3, pp. 885–916, 2018. [Online]. Available: https://doi.org/10.1007/s00145-017-9273-9

[14] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," in *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, ser. Lecture Notes in Computer Science, C. Cid and M. J. J. Jr., Eds., vol. 11349. Springer, 2018, pp. 347–368. [Online]. Available: https://doi.org/10.1007/978-3-030-10970-7_16

[15] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *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*, ser. Lecture Notes in Computer Science, T. Takagi and T. Peyrin, Eds., vol. 10624. Springer, 2017, pp. 409–437. [Online]. Available: https://doi.org/10.1007/978-3-319-70694-8_15

[16] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.

[17] J. Cho, J. Ha, S. Kim, B. Lee, J. Lee, J. Lee, D. Moon, and H. Yoon, "Transciphering framework for approximate homomorphic encryption," in *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*, ser. Lecture Notes in Computer Science, M. Tibouchi and H. Wang, Eds., vol. 13092. Springer, 2021, pp. 640–669. [Online]. Available: https://doi.org/10.1007/978-3-030-92078-4_22

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

[19] C. Dobraunig, M. Eichlseder, L. Grassi, V. Lallemand, G. Leander, E. List, F. Mendel, and C. Rechberger, "Rasta: A cipher with low anddepth and few ands per bit," in *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, ser. Lecture Notes in Computer Science, H. Shacham and A. Boldyreva, Eds., vol. 10991. Springer, 2018, pp. 662–692. [Online]. Available: https://doi.org/10.1007/978-3-319-96884-1_22

[20] C. Dobraunig, M. Eichlseder, T. Korak, S. Mangard, F. Mendel, and R. Primas, "SIFA: exploiting ineffective fault inductions on symmetric cryptography," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2018, no. 3, pp. 547–572, 2018. [Online]. Available: https://doi.org/10.13154/tches.v2018.i3.547-572

[21] C. Dobraunig, L. Grassi, L. Helminger, C. Rechberger, M. Schofnegger, and R. Walch, "Pasta: A case for hybrid homomorphic encryption," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2023, no. 3, pp. 30–73, 2023. [Online]. Available: https://doi.org/10.46586/tches.v2023.i3.30-73

[22] J. Dutertre, T. Riom, O. Potin, and J. Rigaud, "Experimental analysis of the laser-induced instruction skip fault model," in *Secure IT Systems - 24th Nordic Conference, NordSec 2019, Aalborg, Denmark, November 18-20, 2019, Proceedings*, ser. Lecture Notes in Computer Science, A. Askarov, R. R. Hansen, and W. Rafnsson, Eds., vol. 11875. Springer, 2019, pp. 221–237. [Online]. Available: https://doi.org/10.1007/978-3-030-35055-0_14

[23] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, p. 144, 2012. [Online]. Available: http://eprint.iacr.org/2012/144

[24] A. Feldmann, N. Samardzic, A. Krastev, S. Devadas, R. Dreslinski, K. Eldefrawy, N. Genise, C. Peikert, and D. Sanchez, "F1: A fast and programmable accelerator for fully homomorphic encryption (extended version)," 2021.

[25] R. Fernàndez-València, "Verifiable encodings in multigroup fully homomorphic encryption," *CoRR*, vol. abs/2303.08432, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2303.08432

[26] Accessed in April, 2024 from the official PQC source codes for Kyber and Dilithium, based on the public domain implementation in crypto_hash/keccakc512/simple/ from http://bench.cr.yp.to/supercop.html by Ronny Van Keer and the public domain "TweetFips202" implementation from https://twitter.com/tweetfips202 by Gilles Van Assche, Daniel J. Bernstein, and Peter Schwabe.

[27] R. Geelen, M. Van Beirendonck, H. V. L. Pereira, B. Huffman, T. McAuley, B. Selfridge, D. Wagner, G. Dimou, I. Verbauwhede, F. Vercauteren, and D. W. Archer, "BASALISC: Flexible Asynchronous Hardware Accelerator for Fully Homomorphic Encryption," 2022. [Online]. Available: https://arxiv.org/abs/2205.14017

[28] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, USA, 2009. [Online]. Available: https://searchworks.stanford.edu/view/8493082

[29] ——, "Computing arbitrary functions of encrypted data," *Commun. ACM*, vol. 53, no. 3, pp. 97–105, 2010. [Online]. Available: https://doi.org/10.1145/1666420.1666444

[30] A. K. Ghosal, A. Sardar, and D. Roychowdhury, "Differential fault analysis attack-tolerant hardware implementation of AES," *J. Supercomput.*, vol. 80, no. 4, pp. 4648–4681, 2024. [Online]. Available: https://doi.org/10.1007/s11227-023-05632-2

[31] S. Ghosh, D. Saha, A. Sengupta, and D. R. Chowdhury, "Preventing fault attacks using fault randomization with a case study on AES," in *Information Security and Privacy - 20th Australasian Conference, ACISP 2015, Brisbane, QLD, Australia, June 29 - July 1, 2015, Proceedings*, ser. Lecture Notes in Computer Science, E. Foo and D. Stebila, Eds., vol. 9144. Springer, 2015, pp. 343–355. [Online]. Available: https://doi.org/10.1007/978-3-319-19962-7_20

[32] J. Guo, T. Peyrin, and A. Poschmann, "The PHOTON family of lightweight hash functions," in *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, ser. Lecture Notes in Computer Science, P. Rogaway, Ed., vol. 6841. Springer, 2011, pp. 222–239. [Online]. Available: https://doi.org/10.1007/978-3-642-22792-9_13

[33] J. Guo, T. Peyrin, A. Poschmann, and M. J. B. Robshaw, "The LED block cipher," in *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, ser. Lecture Notes in Computer Science, B. Preneel and T. Takagi, Eds., vol. 6917. Springer, 2011, pp. 326–341. [Online]. Available: https://doi.org/10.1007/978-3-642-23951-9_22

[34] J. Ha, S. Kim, W. Choi, J. Lee, D. Moon, H. Yoon, and J. Cho, "Masta: An he-friendly cipher using modular arithmetic," *IEEE Access*, vol. 8, pp. 194 741–194 751, 2020. [Online]. Available: https://doi.org/10.1109/ACCESS.2020.3033564

[35] J. Ha, S. Kim, B. Lee, J. Lee, and M. Son, "Rubato: Noisy ciphers for approximate homomorphic encryption," in *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*, ser. Lecture Notes in Computer Science, O. Dunkelman and S. Dziembowski, Eds., vol. 13275. Springer, 2022, pp. 581–610. [Online]. Available: https://doi.org/10.1007/978-3-031-06944-4_20

[36] S. Halevi and V. Shoup, "Faster homomorphic linear transformations in helib," in *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, ser. Lecture Notes in Computer Science, H. Shacham and A. Boldyreva, Eds., vol. 10991. Springer, 2018, pp. 93–120. [Online]. Available: https://doi.org/10.1007/978-3-319-96884-1_4

[37] M. Hell, T. Johansson, A. Maximov, W. Meier, and H. Yoshida, "Grain-128aeadv2: Strengthening the initialization against key reconstruction," in *Cryptology and Network Security - 20th International Conference, CANS 2021, Vienna, Austria, December 13-15, 2021, Proceedings*, ser. Lecture Notes in Computer Science, M. Conti, M. Stevens, and S. Krenn, Eds., vol. 13099. Springer, 2021, pp. 24–41. [Online]. Available: https://doi.org/10.1007/978-3-030-92548-2_2

[38] H. Hirata, D. Miyahara, V. Arribas, Y. Li, N. Miura, S. Nikova, and K. Sakiyama, "All you need is fault: Zero-value attacks on AES and a new λ-detection m&m," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2024, no. 1, pp. 133–156, 2024. [Online]. Available: https://doi.org/10.46586/tches.v2024.i1.133-156

[39] W. Jung, E. Lee, S. Kim, J. Kim, N. Kim, K. Lee, C. Min, J. H. Cheon, and J. H. Ahn, "Accelerating fully homomorphic encryption through architecture-centric analysis and optimization," *IEEE Access*, vol. 9, pp. 98 772–98 789, 2021. [Online]. Available: https://doi.org/10.1109/ACCESS.2021.3096189

[40] C. Kim, J. Y. Kim, and D. G. Wei, "Invertibility probability of binary matrices team," 2008. [Online]. Available: https://api.semanticscholar.org/CorpusID:15037510

[41] S. Kim, J. Kim, M. J. Kim, W. Jung, J. Kim, M. Rhu, and J. H. Ahn, "BTS: An Accelerator for Bootstrappable Fully Homomorphic Encryption," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 711–725. [Online]. Available: https://doi.org/10.1145/3470496.3527415

[42] D. Le, S. L. Yeo, and K. Khoo, "Algebraic differential fault analysis on SIMON block cipher," *IACR Cryptol. ePrint Arch.*, p. 436, 2021. [Online]. Available: https://eprint.iacr.org/2021/436

[43] B. Li and D. Micciancio, "On the security of homomorphic encryption on approximate numbers," in *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, ser. Lecture Notes in Computer Science, A. Canteaut and F. Standaert, Eds., vol. 12696. Springer, 2021, pp. 648–677. [Online]. Available: https://doi.org/10.1007/978-3-030-77870-5_23

[44] Z. Li, S. D. Galbraith, and C. Ma, "Preventing adaptive key recovery attacks on the gsw levelled homomorphic encryption scheme," in *Provable Security*, L. Chen and J. Han, Eds. Cham: Springer International Publishing, 2016, pp. 373–383.

[45] P. Longa and M. Naehrig, "Speeding up the number theoretic transform for faster ideal lattice-based cryptography," in *Cryptology and Network Security*, S. Foresti and G. Persiano, Eds. Cham: Springer International Publishing, 2016, pp. 124–139.

[46] P. Méaux, C. Carlet, A. Journault, and F. Standaert, "Improved filter permutators for efficient FHE: better instances and implementations," in *Progress in Cryptology - INDOCRYPT 2019 - 20th International Conference on Cryptology in India, Hyderabad, India, December 15-18, 2019, Proceedings*, ser. Lecture Notes in Computer Science, F. Hao, S. Ruj, and S. S. Gupta, Eds., vol. 11898. Springer, 2019, pp. 68–91. [Online]. Available: https://doi.org/10.1007/978-3-030-35423-7_4

[47] P. Méaux, A. Journault, F. Standaert, and C. Carlet, "Towards stream ciphers for efficient FHE with low-noise ciphertexts," in *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*, ser. Lecture Notes in Computer Science, M. Fischlin and J. Coron, Eds., vol. 9665. Springer, 2016, pp. 311–343. [Online]. Available: https://doi.org/10.1007/978-3-662-49890-3_13

[48] A. Menu, J. Dutertre, O. Potin, J. Rigaud, and J. 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*. IEEE, 2020, pp. 1–7. [Online]. Available: https://doi.org/10.1109/DTIS48698.2020.9081261

[49] A. C. Mert, Aikata, S. Kwon, Y. Shin, D. Yoo, Y. Lee, and S. S. Roy, "Medha: Microcoded Hardware Accelerator for computing on Encrypted data," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2023, no. 1, pp. 463–500, 2023. [Online]. Available: https://doi.org/10.46586/tches.v2023.i1.463-500

[50] D. Micciancio, "A first glimpse of cryptography's holy grail," *Commun. ACM*, vol. 53, no. 3, p. 96, 2010. [Online]. Available: https://doi.org/10.1145/1666420.1666445

[51] D. Mukhopadhyay, "An improved fault based attack of the advanced encryption standard," in *Progress in Cryptology - AFRICACRYPT 2009, Second International Conference on Cryptology in Africa, Gammarth, Tunisia, June 21-25, 2009. Proceedings*, ser. Lecture Notes in Computer Science, B. Preneel, Ed., vol. 5580. Springer, 2009, pp. 421–434. [Online]. Available: https://doi.org/10.1007/978-3-642-02384-2_26

[52] M. Naehrig, K. E. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011*, C. Cachin and T. Ristenpart, Eds. ACM, 2011, pp. 113–124. [Online]. Available: https://dl.acm.org/citation.cfm?id=2046682

[53] G. B. J. D. M. Peeters and G. V. Assche, "The Keccak reference," Round 3 submission to NIST SHA-3, 2011. [Online]. Available: http://keccak.noekeon.org/Keccak-reference-3.0.pdf

[54] R. Radheshwar, M. Kansal, P. Méaux, and D. Roy, "Differential Fault Attack on Rasta and FiLIP_{DSM}," *IEEE Trans. Computers*, vol. 72, no. 8, pp. 2418–2425, 2023. [Online]. Available: https://doi.org/10.1109/TC.2023.3244629

[55] D. Roy, B. N. Bathe, and S. Maitra, "Differential fault attack on kreyvium & FLIP," *IEEE Trans. Computers*, vol. 70, no. 12, pp. 2161–2167, 2021. [Online]. Available: https://doi.org/10.1109/TC.2020.3038236

[56] M. 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*. IEEE Computer Society, 2006, pp. 487–496. [Online]. Available: https://doi.org/10.1109/FOCS.2006.52

[57] D. Saha, S. Kuila, and D. R. Chowdhury, "Escape: Diagonal fault analysis of APE," in *Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India, New Delhi, India, December 14-17, 2014, Proceedings*, ser. Lecture Notes in Computer Science, W. Meier and D. Mukhopadhyay, Eds., vol. 8885. Springer, 2014, pp. 197–216. [Online]. Available: https://doi.org/10.1007/978-3-319-13039-2_12

[58] D. Saha, D. Mukhopadhyay, and D. R. Chowdhury, "A diagonal fault attack on the advanced encryption standard," *IACR Cryptol. ePrint Arch.*, p. 581, 2009. [Online]. Available: http://eprint.iacr.org/2009/581

[59] X. M. Saß, R. Mitev, and A. Sadeghi, "Oops..! I glitched it again! how to multi-glitch the glitching-protections on ARM trustzone-m," in *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, J. A. Calandrino and C. Troncoso, Eds. USENIX Association, 2023, pp. 6239–6256. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/sass

[60] D. Sprenkels, "The Kyber/Dilithium NTT," https://dsprenkels.com/ntt.html.

[61] M. Tunstall, D. Mukhopadhyay, and S. Ali, "Differential fault analysis of the advanced encryption standard using a single fault," in *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*, ser. Lecture Notes in Computer Science, C. A. Ardagna and J. Zhou, Eds., vol. 6633. Springer, 2011, pp. 224–233. [Online]. Available: https://doi.org/10.1007/978-3-642-21040-2_15

[62] ——, "Differential fault analysis of the advanced encryption standard using a single fault," in *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*, ser. Lecture Notes in Computer Science, C. A. Ardagna and J. Zhou, Eds., vol. 6633. Springer, 2011, pp. 224–233. [Online]. Available: https://doi.org/10.1007/978-3-642-21040-2_15

[63] A. Viand, C. Knabenhans, and A. Hithnawi, "Verifiable fully homomorphic encryption," *CoRR*, vol. abs/2301.07041, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2301.07041

[64] Z. Xing, Z. Zhang, J. Liu, Z. Zhang, M. Li, L. Zhu, and G. Russello, "Zero-knowledge proof meets machine learning in verifiability: A survey," *CoRR*, vol. abs/2310.14848, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2310.14848

Appendix

AES-GCM is a standardized AEAD scheme used in TLS protocols. It uses the GCM mode along with the AES symmetric-key scheme. The GCM mode is widely adopted in applications for its performance and is not restricted to AES. Any SE scheme can be used to replace AES in this mode to offer AT in the context of HHE. Since AES is a well-known standard, in this section, we will give a brief idea of new SE schemes design. For this, we choose PASTA [21] 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 9, 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 HSD, this key is encrypted and securely transmitted to the server as K_{FHE} (done only once in the beginning). Hence, both K and the client's message (m_i) remain concealed from the server.

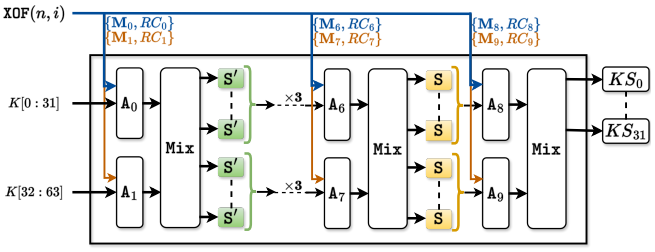

Figure 9. The PASTA-4 permutation (π) takes as input the key (K), nonce (n), and counter (i), and ultimately generates the truncated result- key stream (KS).

With the public and private variables clearly defined, we now describe the PASTA scheme. It operates as a stream cipher and comprises two variants: 3-round PASTA-3 and 4-round PASTA-4. Figure 9 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 (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.

1. MASTA and RASTA design overview

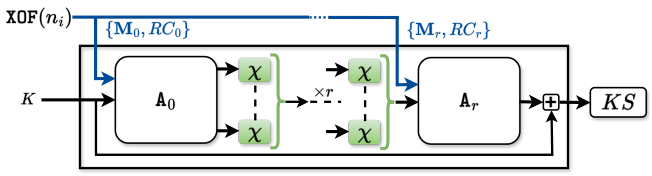

Figure 10. The r-round MASTA permutation. The '+' sign in the box refers to modular vector addition.

MASTA [34] follows a slightly different approach as shown in Figure 10. 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 19). 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 [19] is similar to MASTA with the only difference being operation over $\mathbb{Z}_2$ and the choice of S-box.

$$\mathtt{S}_\chi(X^j) = \begin{cases} X^j \pmod p & \text{if } j \le 1, \\ X^j + X^{j-1} \cdot X^{j-2} \pmod p & \text{otherwise,} \end{cases} \tag{19}$$

2. HERA design overview

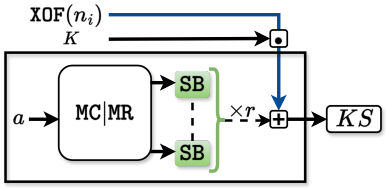

Figure 11. This figure gives a general idea of HERA and RUBATO permutation. The '+' and '·' signs in the boxes refer to modular vector addition and modular vector dot product.

HERA [17] adopts a distinct approach compared to PASTA and MASTA (as shown in Figure 11). 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).

3. RUBATO design overview

RUBATO [35] was developed after PASTA and HERA, utilizing design principles from both schemes (Figure 11). 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.

The matrix multiplication required for processing the affine layers of the new SE schemes is an expensive operation. This is not a problem on the client side, as data can be fetched and stored as desired. Contrarily, on the server side during HSD, data is encoded/encrypted into polynomials, which do not offer much flexibility and consume significant multiplicative depth. To reduce this, the baby-step giant-step method [36] is used [21], [34] for expediting matrix-vector multiplication on encoded/encrypted data, as shown in Section C. 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 $\texttt{rot}_j$ function rotates the input vector by j, and the $\texttt{diag}$ function gives rotated diagonals of $\mathbf{M}$.

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

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.

The KECCAK [53] permutation function is renowned for its role as the underlying component of the secure hashing algorithm standard SHA-3. For the SE 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.

The **Number Theoretic Transform** (NTT) [45], [60] 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.

The steps for generating the invertible matrices ($\mathbf{M}_i$) of PASTA-E_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 20 [32], [33]. This ensures that the matrix is invertible

$$\mathbf{M}_i^{j+1} = \mathbf{M}_i^j \cdot \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ 0 & 0 & 0 & \cdots & 1 \\ \alpha_0 & \alpha_1 & \alpha_2 & \cdots & \alpha_{t-1} \end{bmatrix} \quad \forall\ 1 \leq j < t \tag{20}$$

As stated in Section 4.3, the diffusion in RV_r completely depends on the diffusion obtained from XOF. To understand why the FIP of inducing a fault in Keccak permutation, used for PASTA and MASTA, 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 12.

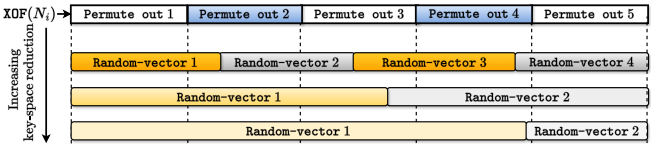

Figure 12. This figure shows that more KECCAK permutations for random vector generation of HERA make it easier to target a specific vector without affecting others, leading to a higher reduction in the keyspace.

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, 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 could have a similar effect, such as a matrix counter skip or skipping the matrix multiplication instruction. This will have a success probability of 1 without any reliance on prime size or rejection rate.

Although attacking the client's device is more realistic and is generally the target of all prior works, we cannot completely rule out the possibility of the fault being induced on the server side by a malicious insider. Hence, informing the community about the potential of such an attack is imperative, acknowledging that its feasibility remains an open debate.

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

$$\Delta E_\pi = E_\pi(K, n_i) - E'_\pi(K, n_i) \pmod p$$

Proof. For all HHE schemes that are stream ciphers (PASTA, SASTA, RASTA, MASTA, HERA, and RUBATO) or use stream modes like the AES-GCM, there is a computation

mirror effect on the client and server side. This implies that the $\mathtt{E}_\pi$ computation is done in the same way on both the server and client side. The only difference being on the server-side the Key is encrypted. Thus, a fault induced during the server-side homomorphic evaluation of SE decryption circuit would result in faulty permutation $\mathtt{E}'_\pi(K, n_i)$. This, coupled with the client's fault-free computation, will result in the desired differential. 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. □

Let us examine the ideal scenario where a fault (non-zero by definition) fully diffuses only to the diagonal of the last matrix/vector utilized in all the constructions discussed above. The steps for generating an invertible matrix are explained in item C. The server-side computation offers a remarkable opportunity for attack surface, thanks to the use of the baby-step giant-step based matrix-vector multiplication method (discussed in Section C). This method encodes the diagonals of matrix $\mathbf{M}$, or the vector in NTT form (discussed in item C). These encoded values 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 to reverse the transform. Since our plaintext is a diagonal/vector, the fault spreads to the whole diagonal/vector. For practically inducing this known fault, an instruction skip fault during the NTT transform is sufficient, as the values are public. Such a fault can be induced with high precision as shown in [9], [12], [22], [48]. However, attacking the server-side computation can be challenging due to limited opportunities and constraints.

Although the fault can be introduced on both the client and server side (Corollary 1), it is more realistic that the attacker gains momentarily access to one device and not both. Hence, although we show interesting attack surfaces on both the client and server sides, we only analyze the schemes under the client-side fault injection points.