SASTA-DFA: Single-Fault Nonce Respecting Differential Attack on Hybrid Homomorphic Encryption

Aikata Aikata[1] Ahaan Dabholkar[2], Dhiman Saha[3] and Sujoy Sinha Roy[1]

[1] Graz University of Technology, Austria
{aikata,sujoy.sinharoy}@tugraz.at
[2] Purdue University, USA
adabholk@purdue.edu
[3] Indian Institute of Technology Bhilai, India
dhiman@iitbhilai.ac.in

Abstract.
Fully Homomorphic Encryption offers an effective solution for privacy-preserving computation, but its adoption is hindered by substantial computational and communication overheads. To address these, the Hybrid Homomorphic Encryption (HHE) protocol was developed, where the client encrypts data using a symmetric encryption scheme (SE), and the server homomorphically evaluates its decryption. Previous studies have demonstrated that the HHE protocol has no impact on the correctness of applications; however, in this work, we shift the focus to its security resilience when subjected to Differential Fault Analysis (DFA). While DFA has proven effective against standalone symmetric-key primitives, no DFA study has been proposed that exploits the HHE protocol as a whole. Furthermore, previous DFA approaches on SE rely on strong assumptions such as nonce reuse, which limits their applicability in real-world protocols or practical applications.
In this work, we show that the structure of the HHE protocol itself exposes new avenues for fault exploitation. We introduce SASTA-DFA, which, to our knowledge, is the first DFA targeting HHE protocol in its entirety. Our study demonstrates that an attacker can achieve complete key recovery with a single fault injection. A key feature of this attack is that it does not require nonce reuse, thus adhering to nonce-related specifications. We adapt the IND-CPA$^{\mathrm{D}}$ threat model proposed by Li and Micciancio at Eurocrypt'21 for HHE in the context of fault attacks.
We conduct the first DFA study on the emerging HHE-specific integer-based SE schemes— RUBATO, HERA, PASTA, and MASTA. Notably, our attack methodology is generalizable and applicable to a broader class of HHE-friendly SE schemes, including boolean schemes like RASTA and even the standard scheme AES. We also present the first experimental validation of fault analysis on these new HHE-enabling schemes. Our attack, mounted on an ATXmega128D4-AU microcontroller, successfully demonstrates full key recovery. Finally, we also extend SASTA-DFA to Authenticated Transciphering protocols under a weaker threat model that removes any functional dependency.

Keywords: DFA · FHE · PASTA · MASTA · HERA · RUBATO · RASTA

1 Introduction

Fully Homomorphic Encryption (FHE) schemes [Gen09, CKKS17, BGV11, CGGI20, FV12, Bra12] offer such privacy preserving capabilities. However, their widespread adoption is

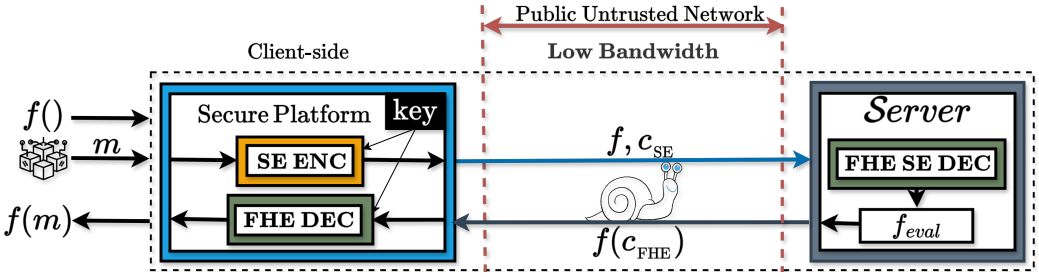

Figure 1: The client symmetrically encrypts the plaintext (SE ENC) and sends it to the server. The server converts it to FHE ciphertext, evaluates the function, and returns the result. The client then decrypts (FHE DEC) the final ciphertext homomorphically.

inhibited by significant computational and communication overheads. This is because FHE encryption schemes (lattice-based) transform data into substantially larger polynomials and rely on costly polynomial arithmetic for performing computations on the encrypted data. While server-side operations can be accelerated with specialized hardware [MAK+23, GVBP+22, AMK+25], such hardware is often unaffordable for end-users or resource-constrained devices, making client-side homomorphic encryption a major bottleneck.

To address these problems, the Hybrid Homomorphic Encryption (HHE) paradigm, also known as *transciphering* [NLV11], was introduced. The main idea behind HHE is to replace the expensive homomorphic encryption methods with symmetric key encryption. In the HHE protocol, illustrated in Fig. 1, clients encrypt their data with a symmetric key scheme (e.g., AES) 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, for example, Neural Network inference. 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 for server-side homomorphic decryption. As a result new HHE tailored Symmetric Encryption (SE) schemes – Pasta [DGH+23], Masta [HKC+20], Hera [CHK+21], and Rubato [HKL+22], have been proposed. Unlike standard schemes, these operate over integers in prime fields ($\mathbb{F}_p$) and have low multiplicative depth, enabling efficient homomorphic symmetric decryption.

Recent works [BBS21, ADE+23] have proposed *Authenticated Transciphering* (AT) to further enhance the capabilities of HHE by ensuring the integrity and authenticity of encrypted data. The motivation is to ensure that the client receives the result on the correct data and not manipulated data, as the ciphertexts can be maliciously altered in communication. They achieve this by using symmetric key schemes (e.g., AES-GCM) in Authenticated Encryption with Associated Data (AEAD) mode. Its ability to obtain a Tag helps the client to validate the data integrity.

1.1 Research Gap

Physical attacks on cryptographic implementations [SMS23, BU02, TBP20, BDL97, BS97, WPH+22] exploit the physical characteristics of the device running the cryptographic primitive to recover secret parameters involved in its computation. These attacks include measuring the output size [BU02], execution time [TBP20], and power consumption [WPH+22] or disturbing a cryptographic computation to recover sensitive data [SMS23, BDL97, BS97]. Physical attacks are often more efficient than mathematical cryptanalytic attacks and are considered a severe threat, especially in applications where attackers can be near the

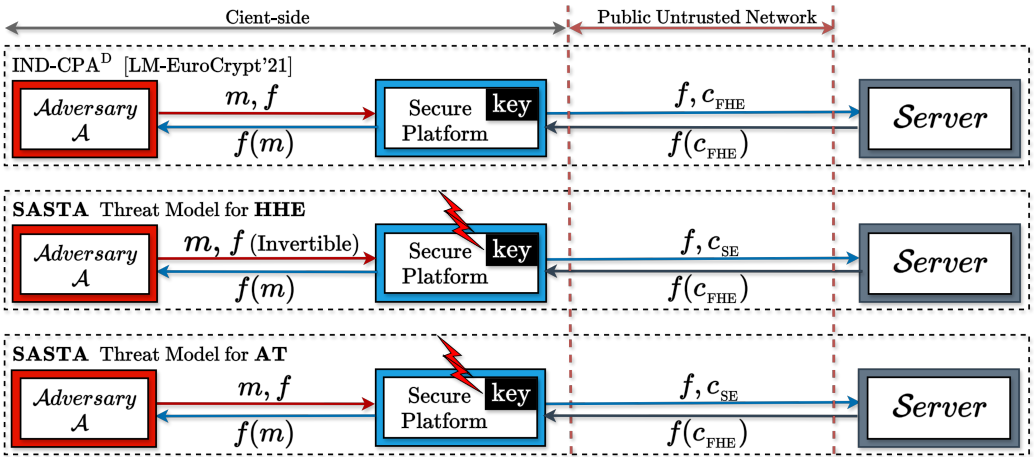

Figure 2: The IND-CPA^D threat model [LM21] and its adaptation to the SASTA-DFA model for HHE and AT, which extends it by incorporating fault injection capability.

computing device. Despite its promise, the HHE protocol itself and the new SE schemes have not undergone extensive cryptanalysis in the context of physical attacks such as Differential Fault Analysis (DFA) [BS97].

To date, there have been few DFA studies [RBM21, RKMR23] on HHE tailored SE schemes [DEG+18, CCF+18, MCJS19, MJSC16]. These studies focus exclusively on the standalone SE primitives without leveraging the specific characteristics of the HHE protocol. Their attack approaches are similar to classical fault attacks on AES, for example [TMA11], and require strong assumptions like nonce reuse. However, this deviates from the scheme specifications and is prohibited by cryptographic protocols like SSL/TLS [GGF22, DEK+16]. This motivates us to investigate novel DFA methodologies tailored for HHE under realistic constraints. Next, prior DFA works target SE schemes operating on boolean data [MJSC16, MCJS19, DEG+18, CCF+18, MR24]. In contrast, emerging SE algorithms for HHE, such as Pasta [DGH+23], Masta [HKC+20], Hera [CHK+21], and Rubato [HKL+22] operate in prime fields. An analysis of these HHE-tailored SE schemes under DFA is unexplored yet much needed due to the vast potential for deployment of privacy-preserving applications [NWH+25]. Finally, note that both HHE and FHE protocols share the same FHE decryption routine and differ only in the encryption. It remains another open challenge to analyze how a fault attack on HHE encryption routine can be used to recover the secret key.

Thus, while prior works have shown that HHE *introduces minimal loss on the correctness of many applications, we shift the spotlight to a more critical question: Security under Differential Fault Analysis.*

1.2 The SASTA Threat Model

Our threat model follows the seminal work of Li and Micciancio [LM21], where the attacker tries to recover the secret key in the setting shown in Fig 2, termed as IND-CPA^D. They show how this setting, where the attacker has access to the results of the restricted decryption oracle, is practically enabled by FHE. The FHE community widely acknowledged this work and actively added fixes/patches for the attack by changing the error distribution during decryption in various libraries like HElib [HS20] and PALISADE [PAL21]. Following this groundbreaking observation, in this work, we aim to explore the same setting in the presence of a fault, as showcased in Fig. 2. Note that since our attack is not dependent on

the error distribution, it cannot be bypassed by the patches used for preventing the attack shown in [LM21]. Thus, our attack remains practical and realistic, and its prevention necessitates additional fault attack countermeasures. We propose a DFA technique that can be applied to both HHE and AT. For HHE, the function to be evaluated can be non-trivial but invertible. However, this assumption is relaxed when we apply Sasta-DFA on AT, and any arbitrary function can be queried, analogous to the original IND-CPA$^{\mathrm{D}}$ setting, as shown in Fig. 2.

Threat Model in Practical Contexts. In practice, Li and Micciancio [LM21] state that the IND-CPA$^{\mathrm{D}}$ can be realized in systems where decryption results are either publicly accessible or selectively shared with specific parties, for example, in cloud computing systems. In such systems, Keys are securely managed by a hardware security module (HSM) or secure enclave, rendering them inaccessible to an attacker present on the client. A real-life example is MindNetworks' deployment [Min], where the FHE Decryption Network decrypts encrypted consensus results into plaintext for hubs that require decryption. Recent works [NWH+25] have reinforced that HHE and AT are emerging as a practical and efficient approach for FHE-based privacy-preserving applications. In this setting, the adversary can inject faults during local symmetric encryption and observe decrypted outputs of cloud-evaluated ciphertexts (e.g., after FHE processing) but cannot directly access encryption or decryption keys.

For instance, this applies to privacy-preserving data sharing and processing services used in Secure Cloud-based Medical Diagnosis [Goo, AI]. The hospital encrypts patient data or images locally using symmetric encryption and sends them to the cloud for privacy-preserving analysis or disease prediction. A malicious insider who gains access to the compromised device for a short window can see input data, inject faults, and observe decrypted outputs while keys remain protected by HSMs. Our attack shows that a single fault in just one query is sufficient to reveal the encryption key, exposing the entire medical record database of the hospital. Similar attacks also apply to Edge-to-Cloud Secure Video Analytics [LNGW21], where surveillance systems encrypt video frames locally before uploading them for cloud analytics. An attacker with access to the edge device may tamper with encryption and observe final outputs but cannot access keys stored in secure hardware. Our attack could reveal the video stream transmitted by the device to unauthorized parties and result in a complete privacy breach.

1.3 Contributions

We present Sasta-DFA, a novel DFA attack on HHE to recover private cryptographic keys of the client device. Our main contributions are three-fold.

① **First protocol-level nonce respecting attack.** We are the first to demonstrate how an attacker can exploit the specific properties of the HHE protocol to achieve complete key recovery with a single fault injection without depending on nonce reuse, thus aligning with the practical constraints in real-world scenarios.

The HHE protocol itself facilitates our efficient attack and is central to it. It helps make the attacks not only realistic but also easily applicable. We further extend our attack and, for the first time, show how Authenticated Transciphering (AT) [ADE+23] becomes vulnerable to DFA by leveraging the SASTA threat model. We present an end-to-end key-recovery firmware experiment to validate this.

② **First DFA of** SE **schemes over** $\mathbb{F}_p$**.** Analyzing the SE schemes defined over $\mathbb{F}_p$ under Sasta-DFA is challenging due to a lack of prior DFA literature on these schemes. Therefore, we develop the first key-recovery attack utilizing Sasta-DFA against emerging HHE tailored SE schemes- Pasta [DGH+23], Masta [HKC+20], Hera [CHK+21], Rubato [HKL+22]. This further extends to Rasta [DEG+18] defined over $\mathbb{Z}_2$. The results

Table 1: Comparison of SASTA-DFA with prior works. The average time for key recovery on a 2GHz clock 4GB RAM processor is reported. † stands for results from [AM11b].

Target Scheme	Security	Field	Nonce Resp.	#Faults	Time	HHE Context	Reference
Flip_{530}	80-bit	$\mathbb{Z}_2$	No	1	39hr	✗	[RBM21]
Kreyvium	128-bit	$\mathbb{Z}_2$		3	5min	✗	[RBM21]
Filip_{430}	80-bit	$\mathbb{Z}_2$		1	751hr	✗	[RKMR23]
RASTA-6	80-bit	$\mathbb{Z}_2$		1	96hr	✗	[RKMR23]
AES-GCM	128-bit	$\mathbb{Z}_2$	**Yes**	1	5s	✓	Section 4.6
AES-GCM	256-bit	$\mathbb{Z}_2$		1	45min†	✓	Section 4.6
RASTA-5	128-bit	$\mathbb{Z}_2$		1	0.11s	✓	Section 4.5
RASTA-6	128-bit	$\mathbb{Z}_2$		1	0.12s	✓	Section 4.5
PASTA-3	128-bit	$\mathbb{F}_p$		1	0.50s	✓	Section 4.1
PASTA-4	128-bit	$\mathbb{F}_p$		1	0.21s	✓	Section 4.1
MASTA-4	128-bit	$\mathbb{F}_p$		1	0.09s	✓	Section 4.2
MASTA-5	128-bit	$\mathbb{F}_p$		1	0.07s	✓	Section 4.2
HERA-5	128-bit	$\mathbb{F}_p$		1	0.005s	✓	Section 4.3
RUBATO	128-bit	$\mathbb{F}_p$		2	0.15s	✓	Section 4.4

are tabulated in Table 1. This also adds new results to the standard DFA literature (Table 7 in Appendix). Our attack and Key-Recovery technique exploit SASTA-DFA but is not limited to this threat model and can also be utilized in traditional DFA settings.

③ **Attack Proof-of-Concept.** All previous fault analyses conducted on HHE-enabling schemes remain theoretical and lack practical implementations. We bridge this important gap and demonstrate our attack using an 8-bit off-the-shelf micro-controller ATXmega128D4-AU set as a target using the ChipWhispererLite CW1173 board. This attack requires just a *single known fault*, resulting in complete key-recovery. It incorporates an AES firmware. Additionally, we have integrated the firmware for the new HHE tailored SE scheme- HERA to validate SASTA-DFA, further amplifying the scope of our investigation. Our attack implementation and key recovery scripts are open source and available at https://anonymous.4open.science/r/SASTA/. We also comprehensively analyze potential protocol as well as implementation strategies to strengthen the security of the HHE protocol against SASTA-DFA.

- Connection between FHE and SASTA-DFA.

Our key observation highlights a significant vulnerability in the HHE protocol, which makes it susceptible to single-fault DFA even in a realistic *nonce-respecting* setting. FHE offers strong privacy guarantees, and since the HHE protocol is built on top of FHE, it should maintain these guarantees. However, SASTA-DFA results in full key recovery, thus leading to a loss of privacy offered by the underlying FHE scheme. This underscores the importance of robust protocol design using FHE and secure implementations to protect against physical attacks.

- Design Considerations.

SASTA-DFA bypasses PASTA's truncation, MASTA and RASTA's final-add-key step, and HERA's add-round-key, to recover the secret key. While RUBATO combines add-round-key and truncation for added security, its insufficient truncation and reliance on noise make it vulnerable to our attack. Our findings suggest combining truncation with key-whitening and using multiple independent keys can strengthen security against single-fault attacks by design. This discussion is further detailed in Sec. 7.

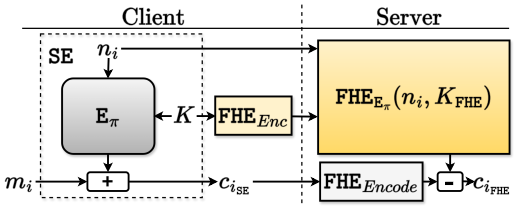

Figure 3: Workflow of HHE is illustrated here. The homomorphically encrypted key ($K_{\mathtt{FHE}}$) is communicated only once. After this, multiple small ciphertexts ($c_{i_{\mathtt{SE}}}$) encrypted using SE are sent along with nonce. The server performs a 'homomorphic SE decryption circuit evaluation' (HSD) and obtains a homomorphically encrypted ciphertext $c_{i_{\mathtt{FHE}}}$. Here $\mathtt{E}_\pi$ refers to any SE permutation (e.g., AES), and i refers to i-th iteration of the HHE protocol.

Paper Outline.

In Sec. 2, we briefly describe HHE and AT. We also detail HHE enabling SE scheme-Pasta's routines and provide more background on prior DFA works. In Sec. 3, we introduce the threat model and show how a differential can be obtained for HHE and AT protocols. Sec. 4 shows how the differentials with carefully selected fault injection points can result in key recovery for various schemes. In Sec. 5, we show the experimental evaluation of the attack on a microcontroller and how it results in complete key recovery in real time. The countermeasures for the attack are discussed in Sec. 6. Sec. 7 discusses the implication on design decisions, limitations and future scope of the work, and Sec. 9 concludes the work.

2 Background and Related Work

Notation: $\mathbb{Z}_2$ refers to Boolean values and $\mathbb{F}_p$ 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 and $\mathbf{M}^i$ for rows in a matrix. Variables with an apostrophe sign or red colour (e.g., $c', C', \mathbf{M}'$) refer to faulty values resulting from fault injection. The subscripts FHE or SE imply the value is encrypted using FHE or SE.

2.1 Homomorphic Encryption

Before we delve into the specifics of 'Hybrid' Homomorphic Encryption, we briefly introduce Homomorphic Encryption. The first FHE scheme was introduced by Gentry in 2009 [Gen09], and since then, several FHE schemes have been proposed by researchers, such as BGV [BGV11], BFV [FV12, Bra12], CKKS [CKKS17, CHK+18], and TFHE [CGGI20]. Most FHE schemes rely on complex mathematical problems like Learning With Errors (LWE) or its faster polynomial ring variant, Ring Learning With Errors (RLWE). However, a significant drawback is that they transform plaintext data into much larger polynomials when homomorphically encrypted, resulting in substantial communication and computational overhead, often ranging from $10,000\times$ to $100,000\times$ [JLK+21].

2.2 Hybrid Homomorphic Encryption (HHE)

HHE addresses the above issues, and a detailed top-level overview of the HHE protocol is depicted in Fig. 3, and Table 2. It proceeds as follows.

Table 2: A simplified algorithmic description of the HHE protocol. Here, i refers to i-th iteration of the HHE protocol.

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

- 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. 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 $\texttt{FHE}_{Encrypt}$ 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 $c_{i_{\texttt{SE}}} = m_i + \texttt{E}_\pi$ to the server along with the public nonce n_i. $\texttt{E}_\pi$ refers to the key-dependent permutation that results in a keystream, which is added to plaintext for encryption.

- The server uses the encrypted key $\texttt{K}_{\texttt{FHE}}$ along with the nonce n_i to evaluate the SE decryption (HSD) circuit homomorphically ($\texttt{FHE}_{\texttt{E}_\pi}$). This results in a ciphertext ($c_{i_{\texttt{FHE}}}$), which is only homomorphically encrypted under key sK. This can now be stored on the server or processed as the client desires.

- The client can retrieve the resultant ciphertext and use sK for decryption.

2.3 Authenticated Transciphering (AT)

In the homomorphic setting, AT [BBS21, ADE+23] 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 and 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 ($K_{\texttt{FHE}}$). 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.

[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 do not play a key role in our investigation.

Table 3: A simplified algorithmic description of the Authenticated Transciphering protocol [ADE+23, BBS21].

Client			Server
Initialization			
Generate K, $\{sK, pK\}$ $K_{FHE} = \texttt{FHE}_{Enc}(K)_{pK}$ Send pK and K_{FHE}		$\rightarrow$	Save pK and K_{FHE}
Encryption using SE			
$c_{i_{SE}} = m_i + \texttt{E}_\pi(K, n_i)$ $\texttt{T}_i = GCM(c_{i_{SE}}, K)$ Send $c_{i_{SE}}$, n_i, and $f()$		$\rightarrow$	$c_{i_{SE}}$, n_i, and $f()$
Homomorphic SE *Decryption Circuit Evaluation*			
			$c_{i_{FHE}} = c_{i_{SE}} - \texttt{FHE}_{E_\pi}(n_i, K_{FHE})$ $\texttt{T}_{i_{FHE}} = GCM(c_{i_{SE}}, K_{FHE})$
Function Evaluation			
			Compute $d_{i_{FHE}} = f(c_{i_{FHE}})$
$d_{i_{FHE}}, \texttt{T}_{i_{FHE}}$		$\leftarrow$	Send $d_{i_{FHE}}, \texttt{T}_{i_{FHE}}$
Result Decryption			
If $\texttt{T}_i = \texttt{FHE}_{Dec}(\texttt{T}_{i_{FHE}})_{sK}$:	▶*Decryption Oracle*		
$f(m_i) = \texttt{FHE}_{Dec}(d_{i_{FHE}})_{sK}$	▶*Decryption Oracle*		

First, AT was analyzed by the authors in [BBS21]. They utilized Grain128-AEAD [HJM+21], and this was converted to a TFHE [CGGI20] ciphertext on the server side. This is followed by a recent work [ADE+23], where the authors report benchmarks for authenticated transciphering using standard symmetric encryption schemes such as AES-GCM and ASCON for CKKS [CKKS17] 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. In summary, AT aims to generate an additional authentication tag to ensure data integrity and authenticity.

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. The works [RBM21, RKMR23] make an even stronger assumption by allowing repetition of nonces to mount effective attacks. As pointed out by the authors in [DEK+18], repeating nonces always incurs a certain loss of security, and the inability to repeat the nonce renders conventional DFA techniques infeasible [SKC14]. The authors in [DEK+16] consider a nonce-respecting threat model for AES-GCM but require multiple encryption queries while necessitating that the client not realize the tag is incorrect and refresh the key. This is a strong assumption for FHE use cases.

The susceptibility of classical SE schemes to fault attacks has been extensively analyzed. Several key-recovery attacks have been demonstrated for RSA-CRT Sig-

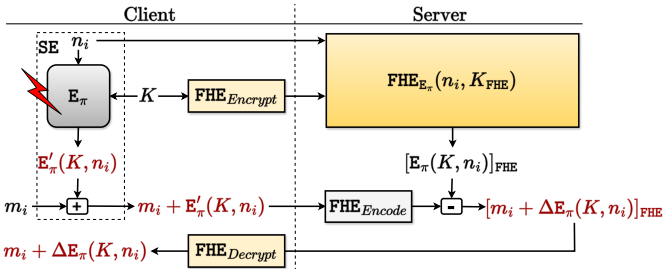

Figure 4: Utilizing SASTA-DFA for differential fault analysis of HHE tailored SE schemes.

natures [BDL97], AES [Muk09, TMA11, HMA+24, GSR24, AM11a, BBB+10]. Prior works [SMC09, TMA11, AM11a] show how even a single fault during AES encryption can significantly reduce the key space, which can be made unique using known plaintext ciphertext pairs. We consider DFA over other attack techniques, such as Statistical Ineffective Fault Attack (SIFA) [DEK+18] or Fault Template Attacks (FTA), because of the HHE setting. The Symmetric Key used in the HHE context is ephemeral and can be refreshed more often than in the traditional Symmetric Encryption setting. This is a problem for techniques like SIFA or FTA that rely on many faulty and fault-free traces under the same key. Moreover, there has been no SIFA or FTA analysis of the new HHE-enabling schemes. Therefore, whether such attacks are even feasible on the new schemes is unknown.

3 The SASTA-DFA Fault Attack Strategy

We adapt the standard DFA framework widely used in prior works [BS97, LYK21, Muk09, TMA11, RBM21] to what we term as SASTA in the context of HHE. While these prior DFA works have made valuable contributions under well-defined models, they assume an adversary capable of replaying the nonce— a very strong assumption that, while analytically useful, may not reflect practical deployment settings. Thus, we investigate whether such assumptions can be eliminated by leveraging the structure of the HHE protocol. We show that without relying on nonce reuse, the protocol's natural behaviour can be exploited to mount a successful attack under a much weaker and practical threat model- SASTA (detailed in Sec. 1.2). Finally, we restrict the attacker to only observe a single query-response pair to mount a successful attack, which is not yet feasible using other nonce-respecting fault attacks (e.g., SIFA, FTA). These relaxed assumptions make our threat model significantly more practical and weaker. Importantly, our paper is the first to explore fault attack surfaces at the HHE *protocol* level of real-life FHE deployment.

3.1 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 (Fig. 4). 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.** 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 decryption oracle, and this output is different from the output corresponding to a fault-free message, thus forming

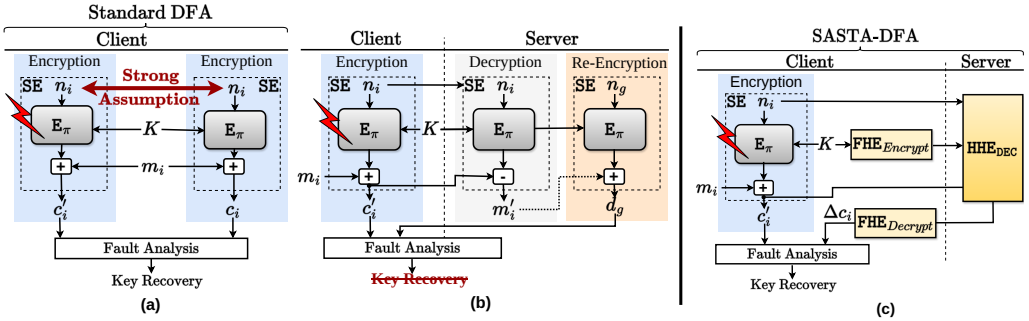

Figure 5: The diagram illustrates the difference between the standard DFA model (a) and the proposed SASTA-DFA model (c), showcasing how the classical setting relies on nonce-reuse, unlike SASTA-DFA. It also presents a case for standard DFA with server interaction (b), indicating how key recovery is infeasible as per specification, $n_i \neq n_g$.

a differential (Table 4 (b)). This process is further elaborated in Sec. 3.3. The attacker observes this *differential* obtained in the output returned by the decryption oracle.

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

In the case of AT, the fault is induced during the very first round function ($\mathtt{E}_\pi$) evaluation (Fig. 6), which is needed only for Tag computation. Thus, the differential is obtained using the valid tag sent by the decryption oracle, which is sent to it by the server in homomorphically encrypted form. This differential is subsequently employed for the purpose of key recovery *without any reliance on the function evaluated.*

3.2 How is SASTA-DFA different from standard DFA?

To create an exploitable differential in a standard DFA setting, an attacker, according to prior works, needs to query the client device twice with the same plaintext while injecting faults during one of the executions. However, this requirement alone is insufficient as a nonce is typically used to ensure that each encryption operation is unique. Thus, to utilize standard DFA, attackers in prior works need to rely on strong nonce-reuse assumptions for key recovery, as shown in Fig. 5 (a). This is illustrated in the following example.

Consider a client-server interaction as shown in Fig. 5 (b). In the symmetric setting, the client encrypts and sends the ciphertext to the server, and the server decrypts it and performs the evaluation requested by the client. The result is then re-encrypted and sent back to the client. Since the nonce (n_i) used for the client's encryption differs from that used for the server's re-encryption (n_g), a fault in the initial client-side encryption (c'_i) for standard DFA will not be exploitable to obtain a differential post re-encryption (d_g). The same applies to a data fetch operation, where the server does not perform evaluation but still performs re-encryption as per the protocol.

Due to these constraints in practical applications, all prior works [RBM21, RKMR23] treat HHE-enabling schemes as standalone symmetric key schemes and assume nonce-reuse to enable DFA. However, nonce-reuse is not permitted in real-world protocols (e.g. SSL/TLS) as it is against the specification (and can lead to replay attacks [GGF22]). Furthermore, as using a nonce is a part of the scheme description, assuming its reuse could also lead to a breakdown of their theoretical security guarantees. Hence, this represents a strong assumption in the standard DFA threat model.

Key Insight. Our key observation is that, by considering the HHE protocol as a whole, we can relax the strong nonce-reuse assumption. As depicted in Fig. 5 (c), the clients send the encrypted data to the server, and the server performs an SE decryption homomorphically. This ciphertext is evaluated and sent to the decryption oracle as part of the FHE protocol. This specific feature enables our attack and overcomes the limitations of prior works using standard DFA. Even if a re-encryption is done (by adding a new $\mathtt{E}_\pi$), the differential stays preserved and does not change post FHE decryption. The re-encryption will only result in the decryption oracle computing additional symmetric decryption to remove the added new $\mathtt{E}_\pi$. Thus, SASTA enables an attacker to perform key recovery without replaying plaintext or any other public parameter, including the nonce.

3.3 Attack on HHE

In HHE, the client device simply performs SE encryption (Table 2) 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 Fig. 3. For schemes defined over $\mathbb{Z}_2$, instead of modular addition/subtraction, `XOR` is used.

The attacker induces a fault in the SE encryption routine (Fig. 4) and uses the output returned by the decryption oracle to generate a differential for key-recovery, as detailed in Table 4. The variable 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 for schemes over $\mathbb{Z}_2$). The subscripts *SE* and *FHE* are removed for simplicity. If a fault occurs during the round function computation ($\mathtt{E}_\pi$) in encryption, the resulting ciphertext c'_i with t coefficients can be expressed as follows.

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

The faulty ciphertext c'^j_i is then sent to the server. The server-side HSD results in a faulty message m'_i, calculated as follows.

$$\begin{aligned} m'^j &= m^j + \mathtt{E}'_\pi(K, n_i)^j - \mathtt{E}_\pi(K, n_i)^j \pmod p \ \forall\ 0 \leq j < t \\ \Delta\mathtt{E}_\pi &= \mathtt{E}'_\pi(K, n_i) - \mathtt{E}_\pi(K, n_i) \pmod p \end{aligned}$$

Since the original message (m_i) is known, and the attacker can choose any invertible function $f()$, the differential ($\Delta\mathtt{E}_\pi$) can be obtained by simply subtracting the known message m_i from m'_i, as shown in Table 4 ©. Thus, when a fault occurs in the round function computation ($\mathtt{E}_\pi$) during encryption, then due to server-side HSD, the fault-induced differential can be obtained *without* nonce-reuse. It can then be utilized for key-recovery, as detailed in Sec. 4.

3.4 Extending SASTA-DFA to AT

As described in Sec. 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 [ADE+23, BBS21], the authors propose AT using schemes on boolean data (e.g., GRAIN128, AES); however, these can also be replaced by the new schemes over finite fields 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-DFA during the very first round function call ($\mathtt{E}_\pi$) of any new encryption since it is only used for tag generation and does not affect the subsequent ciphertext generation (as shown in Fig. 6). A stepwise description is provided in Table 5. This is useful as a lack of dependence on ciphertexts frees us from relying on the knowledge of $f()$, and the attack can work for any arbitrary function or application, for example, private ML inference/training.

Table 4: Faulty computation flow during the HHE protocol. For brevity, we omit the Initialization and Function Evaluation parts of the protocol.

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

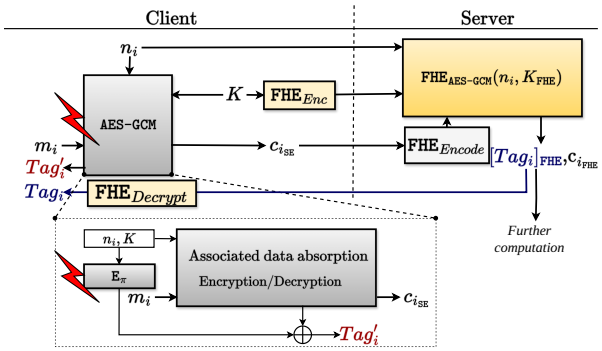

Figure 6: Exploiting SASTA-DFA for Authenticated Transciphering using AES-GCM.

Note that our attack does not alter or influence the associated data absorption and encryption part, shown as a grey box in Fig. 6. Hence, the output (ADE_i-GHASH result) of this computation stays the same in 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. The faulty tag computation can be expressed as follows (Table 5 ⓘ).

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

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 homomorphically encrypted, the server sends it to the client for verification, and the client obtains the original tag from the decryption oracle. Hence, the attacker obtains a differential ($\Delta\mathtt{Tag}$) without requiring nonce-repetition, thus conforming to our threat model (shown in Table 5 ⓘⓘ).

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

The obtained differential can subsequently be used for key-recovery. Observe that the differential obtained here is the same as that for plain HHE. Therefore, the same key-recovery techniques can be utilized.

Table 5: Faulty computation flow during Authenticated Transciphering.

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

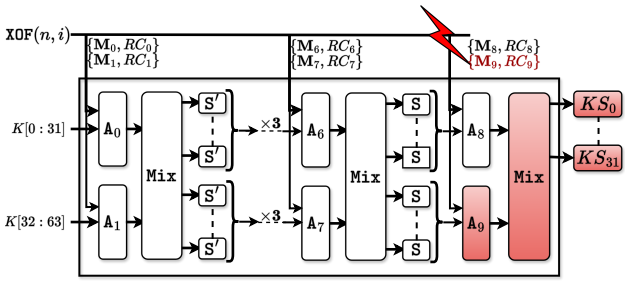

Figure 7: The attack strategy and FIP for PASTA-4 encryption ($KS = \mathtt{E}_\pi(K, n_i)$).

4 Leveraging SASTA-DFA: 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-DFA. We provide detailed case studies for each of the targeted ciphers- AES, PASTA, MASTA, HERA, RUBATO, and RASTA. We identify critical Fault Injection Points (FIPs) which help exploit the differential (Sec. 3.3 or Sec. 3.4) for key-recovery.

4.1 Cracking PASTA's Defenses

The PASTA-4 round function is shown in Fig. 7 (detailed in Appendix B). 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* layer 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 ($KS = \mathtt{E}_\pi(K, n_i)$).

Several candidate FIPs exist for inducing a fault such that it diffuses widely throughout the PASTA-4 encryption (Fig. 15). 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 would not lead to a unique key-recovery, as multiple keys could fulfil the differential condition. This would necessitate multiple such faults to eliminate potential key guesses. Therefore, we seek alternate fault injection points.

We induce a fault in the matrix to be multiplied during step $\mathtt{A}_9$, as depicted in Fig. 7.

If we denote the input state after the last S operation as $X_8||X_9$, the fault-free result is as follows.

$$\mathtt{E}_\pi(K, n_i) = \mathtt{Trunc}(\mathtt{Mix}(\mathtt{A}_8(X_8), \mathtt{A}_9(X_9))) \tag{1}$$
$$\mathtt{E}_\pi(K, n_i) = 2 \cdot (\mathbf{M}_8 \cdot X_8 + RC_8) + (\mathbf{M}_9 \cdot X_9 + RC_9) \tag{2}$$

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

$$\mathtt{E}'_\pi(K, n_i) = 2 \cdot (\mathbf{M}_8 \cdot X_8 + RC_8) + (\mathbf{M}'_9 \cdot X_9 + RC_9) \tag{3}$$
$$\Delta\mathtt{E}_\pi = \Delta\mathbf{M}_9 \cdot X_9 \tag{4}$$
$$X_9 = \Delta\mathbf{M}_9^{-1} \cdot \Delta\mathtt{E}_\pi \tag{5}$$

If $\Delta\mathbf{M}_9$ is known and invertible, X_9 and consequently X_8 can be obtained via Eq. 5 and Eq. 2, respectively. Uniquely recovering $X_8||X_9$ implies full key-recovery of key K. This is because 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 $X_8||X_9$. $\Delta\mathbf{M}_9$ 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 a high probability [KKW08], next we discuss how we choose appropriate fault injection points.

Locating the Achilles Heel (Interesting FIPs). Ensuring $\Delta\mathbf{M}_9$'s invertibility is crucial for determining the appropriate FIP. For this, we look for FIPs that lead to a full matrix diffusion. This is because several studies [Rud06, DS01, KKW08] indicate that the probability of a 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). If the fault induced is fairly random at FIP, with high probability, we can ensure $\Delta\mathbf{M}_9$ has no linear dependencies and is thus invertible as needed for Sasta-DFA.

To induce such a fault, we identify the Keccak (discussed in Appendix F) *permutation* (utilized in XOF) as a viable option. A fault in the XOF function, just before it generates $\mathbf{M}_9$, leads to a full matrix diffusion due to the diffusion property of the Keccak hash function. 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 $\Delta\mathbf{M}_9$. A fault in Keccak that causes $\mathbf{M}_9$ to become faulty will also affect RC_9. Since the value of RC_9 is known, we omit it from our explanation for simplicity.

4.2 Beyond Pasta: Extending the Attack to Masta

The Masta [HKC+20] scheme design was introduced concurrently with Pasta and can be viewed as a direct adaptation of Rasta [DEG+18], originally defined over $\mathbb{Z}_2$, to $\mathbb{F}_p$. As outlined in Appendix C, Masta incorporates two primary functions in its round function: the χ S-box and the affine layer $\mathtt{A}_i$ (illustrated in Fig. 16). The matrix used for multiplication within $\mathtt{A}_i$ is generated in a manner similar to Pasta, ensuring it is invertible by design (Appendix G). In the last round r, if the state with t elements after the last S-box is denoted as X_r, then the final result following the round function 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 $\mathbf{M}_r$ (Fig. 16 Appendix C), and the resulting differential is as presented below.

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

Similar to the case of PASTA, the only unknowns in the above equations are X_r and K. Given that the difference matrix is known and invertible, Eq. 9 can be used to retrieve X_r. No truncation is done here. Instead, the key is added in the end, as shown in Eq. 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 $\Delta\mathbf{M}_r$, similar to the case for PASTA, we can assert that with a high probability (close to 1), $\Delta\mathbf{M}_r$ will be invertible. Hence, MASTA is susceptible to SASTA-DFA, with the key-recovery process significantly simplified due to lack of truncation and the last add-key operation. We also observe that in ciphers with a final add-key operation, the resulting equations are directly expressed in terms of the key. Therefore, in such schemes, the reliance on the invertibility of the matrix $\Delta\mathbf{M}_r$ is significantly reduced, as we discuss next.

Linearization Attack Complexity Analysis. Linearization is one of the most effective attacks on these schemes [DGH+23]. Therefore, it is valuable to examine how, in cases where the matrix $\Delta\mathbf{M}_r$ is not invertible, the available linear constraints contribute to reducing the attack complexity. For a symmetric primitive that admits a polynomial representation of degree d in n_v variables, the total number of possible monomials is given in [DGH+23] as follows:

$$b = \sum_{i=1}^{d} \binom{n_v + i - 1}{i}$$

The linearization attack operates with a time complexity of $\mathcal{O}(b^\omega)$, a memory complexity of $\mathcal{O}(b^2)$, and a data complexity of $\mathcal{O}(b)$. Here, ω is the linear algebra constant, and while typically $2 < \omega \leq 3$, the more conservative and realistic range is $2.37 \leq \omega \leq 2.82$ [GAH+23, LSMI21]. In order to attain 128-bit security, the parameters are chosen to ensure that $\log_2 b / \log_2 b^\omega \leq 64/128$ For MASTA-4/5, $n_v = 128/64$ and $d = 16/32$. In case $\Delta\mathbf{M}_r$ is not invertible and instead has a rank rk, then the number of variables in X_r decrease by rk and hence, using Eq. 7 the total number of unknown key variables also reduce to $n_v - rk$, and the total monomials for linearization reduce as follows.

$$b_{\text{SASTA}} = \sum_{i=1}^{d} \binom{n_v - rk + i - 1}{i} \tag{10}$$

We evaluate that for $\Delta\mathbf{M}_r$ of rank $rk = n_v - 1$, the complexity of the linearization attack is at most 2^{11} for MASTA-4 and 2^{14} for MASTA-5 (Tab. 6), which is significantly below the 128-bit security threshold. Notably, the probability of obtaining a non-invertible differential matrix in the attack on MASTA is negligible, making this a worst-case analysis. Thus, in the worst-case scenario, an attacker could successfully perform a linearization attack for full key recovery. Finally, while our study focuses on 128-bit secure parameters, SASTA-DFA reduces the security margin across all security levels without loss of generality.

4.3 Extending the Attack to HERA

HERA [CHK+21], elaborated in Appendix D, takes a slightly different design approach and uses a pseudo-key-schedule, where the `XOF` output vector (RV) is multiplied with the key

Table 6: The Security bounds (parameters for 128-bit security) before and after SASTA-DFA considering the linearization attack. Considers the case where matrices are not invertible for MASTA, and RASTA, or the key-recovery is incomplete due to insufficient fault diffusion (HERA) or truncation (RUBATO).

Scheme	n_v	d	$\log_2 b$	$\log_2 b^\omega$		$n_v - rk$	$\log_2 b$	$\log_2 \mathrm{b}^\omega_{\mathrm{SASTA}}$	
				$\omega = 2.37$	$\omega = 2.82$		SASTA	$\omega = 2.37$	$\omega = 2.82$
MASTA-4	128	16	69.2	164.1	195.2	1	4	**9.5**	11.3
MASTA-5	64	32	84.6	200.5	238.6	1	5	**11.9**	14.1
HERA-5	16	243	83.3	197.5	235.0	1	7.9	**18.8**	22.3
						4	27.2	64.4	76.1
RUBATO-3†	36	8	29.4	69.8	83.0	4	8.9	**21.2**	25.2
						9	14.6	34.5	41.1
RUBATO-5†	16	32	44.4	105.3	125.3	4	15.8	**37.5**	44.7
						6	21.4	50.7	60.3.4
RASTA-5	525	32	172.9	409.8	487.6	1	5.0	**11.9**	14.0
						4	15.8	37.5	44.7
RASTA-6	351	64	253.1	600.1	714.1	1	6.0	**14.2**	16.8
						4	19.6	46.5	55.4

† The results for RUBATO include the αp factor due to the Gaussian noise.

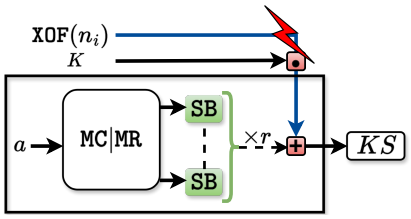

Figure 8: The fault injection points for inducing a fault in HERA or RUBATO permutation before final add-round-key.

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 (Fig. 8) would result in the following equations.

$$\mathtt{E}_\pi(K, n_i) = X_r + RV_r \cdot K \pmod p \tag{11}$$

$$\Delta\mathtt{E}_\pi = RV'_r \cdot K - RV_r \cdot K \pmod p \qquad = \Delta RV_r \cdot K \pmod p \tag{12}$$

In the case of HERA, the known difference is simply a vector ΔRV_r, hence recovering key K, using Eq. 12, only involves field inversions. Thus, the invertibility constraint is relaxed in this setting, and similar to MASTA, no round inversion is required, further simplifying the key-recovery. Furthermore, 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. Appendix H explains how the larger set of FIPs used in previous cases can also be utilized for HERA.

These FIPs impose additional constraints on the attacker's capabilities. To analyze this, we focus on faults injected into the XOF call responsible for generating the final output, RV_r. Our analysis suggests that while a perfect fault diffusion scenario is achievable in the best case, the majority of outcomes result in $rk \in [1, 4]$ values that are not effected, as they were in a previous non-exclusive XOF call. In this case, $n_v - r_k$ key values can be

derived through simple field diversions. For the remaining values, a linearization attack can be applied. The parameters $n_v = 16$ and $d = 243$ for HERA meet the 128-bit security requirements. However, as shown in Tab. 6, when additional equations are obtained from SASTA-DFA (Eq. 4.4), the linearization attack complexity reduces to 2^{19}. This reduction indicates that the attacker's capabilities need not be further enhanced for HERA, as the remaining key space can be recovered using linearization. This attack is particularly effective when the modulus is large, as brute-forcing the remaining keyspace would result in significantly higher complexity.

4.4 The Curious Case of RUBATO

The design of RUBATO [HKL+22] (Appendix E) is heavily influenced by HERA and PASTA. Yet, RUBATO cannot support schemes over integers $\mathbb{F}_p$ 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 noise (e) from Gaussian Distribution $D_{\alpha p}$ ($\alpha p = 10.5, 4.1$ for RUBATO-5,3) at the end of the HHE routine, which modifies the Eq. 11 as follows.

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

After fault injection, the differential is expressed as follows.

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

The e introduced during encryption is not removed during decryption and remains in the data throughout the homomorphic operations. This inherent noise leads to a precision loss, rendering the data unsuitable for precise integer arithmetic and confining its utility to CKKS [CKKS17]. On the other hand, the authors claim that this error enhances the security guarantees, which is why they opt for a low multiplicative depth of two per round and fewer rounds in the scheme. The number of monomials for linearization in the case of RUBATO is stated in [HKL+22] as follows.

$$b = \left[\sum_{i=1}^{d} \binom{n_v + i - 1}{i}\right] \cdot \alpha p$$

The authors in [GAH+23] highlight that the linearization attack security of RUBATO is significantly lower than anticipated. This can be attributed to the scheme's heavy reliance on the added Gaussian noise- e to transform the ciphertext into an LWE-like ciphertext, claiming this as a justification for enhanced security. With SASTA-DFA, additional linear equations involving the key K and the noise e are obtained, as illustrated in Eq. 14. These equations effectively reduce the influence of the added Gaussian noise, thereby decreasing the number of monomials as follows.

$$b_{\text{SASTA}} = \left[\sum_{i=1}^{d} \binom{n_v + i - 1}{i}\right]$$

Thus, the effect of added noise (e) is effectively nullified, and any security assumption solely based on this added noise no longer holds. With $\omega = 2.82$, the linearization attack complexity for RUBATO with 3/5 rounds is $2^{125}/2^{83}$, as shown in Tab. 6. However, after applying SASTA-DFA, this complexity reduces to $2^{115}/2^{77}$ (or $2^{98}/2^{65}$ for $\omega = 2.37$), while eliminating any potential reliance on the added Gaussian noise for security. While this analysis demonstrates that the security assumptions are significantly weakened in the presence of SASTA-DFA, executing this linearization attack in real-time is challenging.

Therefore, for RUBATO specifically, we investigate whether utilizing two differentials instead of just one could enable full key recovery for a computationally constrained adversary. We do not impose any additional constraints or requirements on the two differentials. We obtain two sets of Eq.13 and Eq.14, where, as expected for two distinct encryptions, and everything except the key K changes. From the first differential, we derive a set of possible key values corresponding to the error range. Subsequently, we use the second differential to iterate over these sets and determine the intersection. Remarkably, with this second independent faulty query, we successfully recovered the correct key 99.95% of the time, as confirmed through testing over 100,000 cases. The truncation at the end is minimal, concealing only $rk = 4$ key elements. These remaining elements can be retrieved using trivial linearization with an attack complexity of $2^{21}/2^{38}$ (Tab. 6). Hence, we conclude that while RUBATO marginally increases the attack difficulty, it remains vulnerable under the proposed SASTA-DFA attack model.

4.5 Generalizing SASTA-DFA: SASTA on RASTA

Until now, we explored state-of-the-art HHE schemes over $\mathbb{F}_p$; however, we note that the SASTA-DFA technique is quite general and can also be applied to schemes over $\mathbb{Z}_2$-RASTA [DEG+18], that utilize `XOF` in its construction. The only dependency is that there are much fewer invertible matrices in $\mathbb{Z}_2$ [KKW08] (≈ 0.29). 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.

In this scenario, for the linearization attack, we observe that the rank of the obtained matrix lies within the range $[n_v - 4, n_v]$, as the probability that the matrix has rank at least $n_v - 4$ is ≈ 0.94 [KKW08]. When the rank is n_v, full key recovery is achievable. However, we estimate the linearization attack complexity for a lower rank ($rk = 4$) as $2^{38}/2^{47}$ for RASTA-5/6 (Tab. 6). It is important to note that the key in this case is boolean, so an exhaustive search would have a complexity of 2^4 for the remaining key variables, making it a more efficient attack strategy for RASTA. We 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 Appendix C, RASTA has the similar design principles as MASTA [HKC+20]. The implementation of RASTA also uses KECCAK for `XOF`, thus making it the appropriate exploitable FIP. It results in the desired difference required for SASTA-DFA and leads to a unique key-recovery. Similar to the case of PASTA, fault in Keccak will also affect RC_r. However, its value is known and hence omitted from our explanation.

4.6 Exploiting AES via Known Approaches

Since AES has been a standard for over two decades, numerous studies have analyzed it using DFA [SMC09, AM11b, Muk09, TMA11, HMA+24, GSR24, AM11a, BBB+10]. These works also include both single or multiple-fault-based key-recovery techniques [TMA11, AM11a], [Muk09, HMA+24, GSR24]. Most of these techniques do not even rely on known fault conditions. The FIPs employed in these studies can be utilized to realize SASTA-DFA. Consequently, SASTA-DFA can be employed to extend the same attacks in an HHE setting with weaker assumptions due to non-reliance on nonce-reuse. Note that some DFA methods requiring multiple fault injections also depend on both fault-free and faulty ciphertext pairs for the same nonce. However, with SASTA-DFA, the nonce-reuse assumption is relaxed, making such attacks feasible in practical scenarios.

Summary: Sasta-DFA establishes a universal methodology for DFA across diverse HHE-enabling SE schemes. We elaborated on how Sasta-DFA can be utilized to mount DFA, resulting in full key-recovery on the standard (AES-GCM) and new constructions (Rasta, Pasta, Masta, Hera, and Rubato). This includes both types of HHE-enabling schemes- defined over boolean fields (Rasta, and AES-GCM) as well as those over prime fields (Pasta, Masta, Hera, and Rubato). 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 FIPs on the server-side computations detailed in Appendix I. We emphasize that attacking the client's device is a more realistic setting and is, therefore, the primary target of all prior works on HHE enabling SE schemes.

Next, we present a practical fault injection attack resulting in 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-DFA on one such constrained embedded device, specifically the ATXmega128D4-AU mounted on the ChipWhisperer-Lite CW1173 evaluation board[2]. 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 [SMS23], we also utilize the opensource ChipWhisperer toolchain[3]. Given the limitations of this device, it is difficult to perform an entire conventional client-side FHE computation. However, the use of HHE makes this easily possible[4]. 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, which, to the best of our knowledge, is the first microcontroller realization. This is essential for the development of secure and efficient encryption solutions for a wide range of applications. Next, we discuss how we induce known faults in Keccak and attack the implementation of Hera.

5.1 Fault Injection Technique

The Sasta-DFA attack model does not require a specific fault, and the attack strategy solely relies on the ability to determine the faulty output. Hence, there are no constraints on the type of fault or the specific operation it affects. Several works in literature [SMS23, DRPR19, MDP+20, BFP19, BH22] have validated its feasibility. 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 Sec. 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 [FIPum]. 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 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.

Fig. 9 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

[2] https://rtfm.newae.com/Targets/CW303%20XMEGA/

[3] https://github.com/newaetech/chipwhisperer

[4] The ChipWhisperer toolchain runs AES firmware and supports AES-GCM mode.

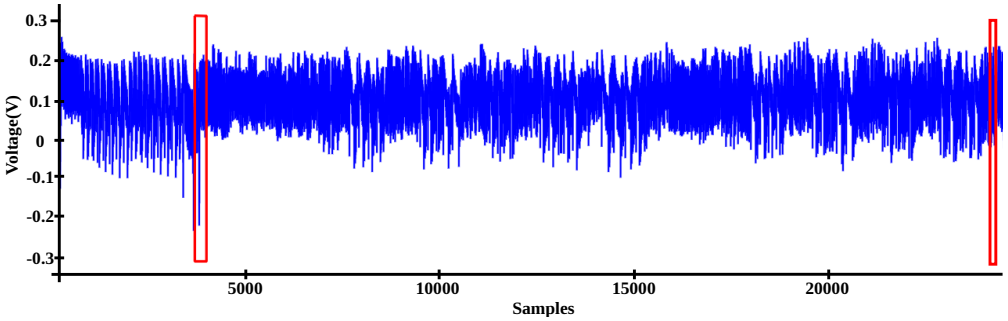

Figure 9: The sample fault injection attack positions targeted in our work. The first position is the beginning of KECCAK permutation, and the second is where the results of permutation are written to the final buffer.

from the preceding permutation to be re-fed. 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 [SMS23] 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.

5.2 Attack and Key-recovery

To showcase a fault injection resulting in full key recovery, we analyze the HERA firmware. As the available interesting FIPs for HERA are a subset of those required for RASTA, PASTA, MASTA, and RUBATO (as discussed in Sec. 4.3) an attack on HERA would also showcase practicality of SASTA-DFA on the other schemes. Fig. 10 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. This skips the initial few KECCAK squeeze instructions, causing the resulting XOF output to shift and provide the necessary fault diffusion. We then employ the strategy discussed in Sec. 4.3 for recovering the secret key of HERA encryption scheme.

We note that our attack does not necessitate power traces shown in Fig. 10. These power traces are merely used to demonstrate the fault injection points and their effect on power consumption. Power analysis attacks are more complex and require specialized and precise equipment than fault attacks to remove the computation noise from the traces. As described in the threat model, our attack only requires an attacker capable of inducing faults during SE encryption.

5.3 Identifying successful fault injection

During our experiments, we identified multiple instances with a zero difference ($\Delta \mathrm{E}_\pi$), which could be the result of the glitch affecting non-critical portions of the code. This shows that not all fault injections yield faulty results and are thus unusable. When a usable fault is induced, our attack requires knowledge of the specific operation affected. Numerous works in the literature [SMS23, DRPR19, MDP+20, BFP19, BH22] have shown

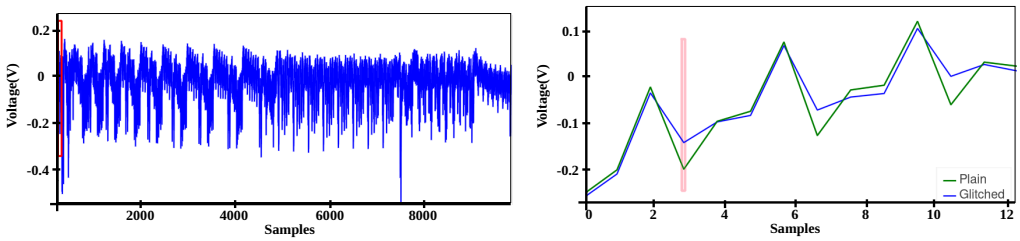

Figure 10: The first plot illustrates fault injection during the last HERA ARK operation. The second plot presents an enlarged view of the normal and faulty computations following the clock glitch. Note that these traces have been extracted from the original plot, and the x-axis is relative.

this to be possible with high precision. We also demonstrated and thus validated this through our fault attack above.

However, in cases where attackers lack precise fault injection techniques, an alternative approach can be employed. They 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 (template), recovering potential faulty outputs and corresponding key guesses. Finally, they can uniquely recover the key with a single fault-free result via post-encryption equality check and elimination.

In conclusion, while the successful execution of the HERA algorithm on the ATXmega128D4-AU highlights the potential of HHE, the SASTA-DFA attack emphasizes the importance of re-evaluating its security guarantees against realistic attack settings. This is crucial to ensure its robustness and effectiveness in real-world scenarios.

5.4 Empirical results for key-recovery

This section details attack specifics and outcomes. First, we examine how the attacker determines the time offset for fault injection. Next, we explore the duration and range of the attack, 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 for a given modulus, 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. Once a fault is induced, we build a template for the possible range of instructions that can be skipped and get a key guess for each. We determine the unique key via the known KS. For our experimental parameters, this template has eight possible instructions (hence, eight guesses at most).

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. An extension to earlier rounds would further increase this range. We show our attack on the last round. This implies countermeasures are only required to protect the last round from our attack. Extending this attack to earlier rounds implies more protection is needed, increasing the cost of countermeasures and, thereby, the severity of the attack. 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 `XOF` phase. An alternative approach is to target a specific instruction. Experimentally, we observe that in the worst case, one out of eight attempts leads to the right (or expected target) fault. The best case is one and an average of four. The average can be lowered

to two using power traces, as the point when the specific XOF computation starts can be approximated more precisely.

Following the attack overview outlined in Sec. 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 utilizes the symmetric-key encryption and FHE decryption part of the HHE protocol.

6 Countermeasure analysis

Pre-generation and Storage. One potential countermeasure against Sasta-DFA involves the offline pre-generation and storage of XOF output. This technique works effectively for Rasta, Hera, and Rubato, as the matrices hold boolean data or the XOF output is a vector. However, this is challenging for Pasta and Masta due to the large amount of XOF output and matrix storage needed (the storage demand for 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.

Redundancy and fault detection. In the long run, redundant computations would be a more suitable countermeasure. The same can be achieved by performing a decryption check on the encrypted data, requiring a complete duplicate computation for key-stream generation. Along with redundancy, we propose employing multiple checks at every stage of the last round for 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.

Infective Countermeasure. Since a stronger adversary can bypass fault detection, infective countermeasure [GSSC15] proves more reliable. Towards this, we employ an XOR-based detection mechanism. The last round is duplicated, and the two duplicate computations are XORed at every stage. After the Affine and Mix transform, this XORed result is ANDed with two random vectors, and the result is XORed back to the two duplicated states. Without 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.

Algorithmic Countermeasures. To protect against the 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-DFA (Sec. 3.4). One performance-hungry way to ensure this is by sending the client's tag also to the server, and it can run an encrypted equality check computation as discussed in [ADE+23]. This incurs a computation overhead on the server side. While the attacker can see the result, it is not exploitable (we need the faulty tag to obtain the differential), thus safeguarding against Sasta-DFA.

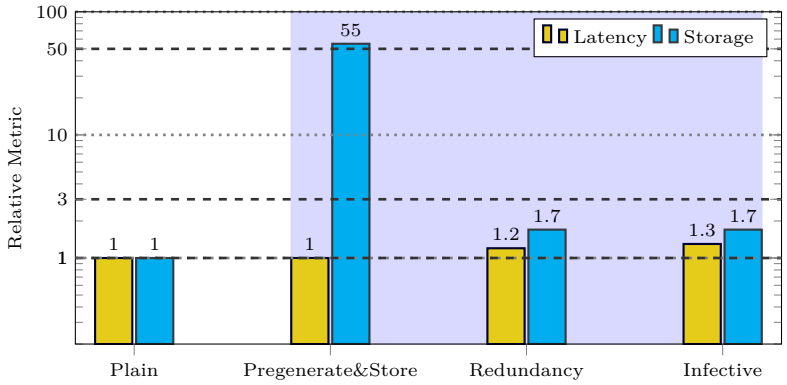

Figure 11: A comparison of latency and storage for different Pasta-4 implementations (with or without countermeasures). The three highlighted implementations are effective against Sasta-DFA. The graph is on a logarithmic scale

6.1 Countermeasure Performance Evaluation

We report performance evaluation results based on the reference implementation provided by the authors of Pasta [DGH+23]. The baseline implementation consumes 274,203 clock cycles on average for 100,000 executions of Pasta-$\mathtt{E}_\pi$. The implementation results are reported for the 12th Gen Intel i7 CPU. In Fig. 11, 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-$\mathtt{E}_\pi$ execution. From Fig. 11, it is evident that the Redundancy and Infective countermeasures offer the best latency vs memory trade-offs.

Do masked implementations resist Sasta-DFA**?** 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. In contrast, the `XOF`, which is the attack target, is not key-dependent and hence will not be masked. Therefore, Sasta-DFA would not be affected by masking. Since no masking scheme is proposed for the new HHE enabling scheme, we only analyze the effect of a possible arithmetic masking.

7 Discussion

7.1 Impact on Design Decisions

Our attack demonstrates that we can bypass the countermeasures employed by various schemes to prevent inversion. For instance, Pasta relies on truncation; however, Sasta-DFA successfully circumvents this, allowing us to recover the original key. Similarly, Masta and Rasta depend on a final Key Add/Xor step for security. Due to this reliance, our attack directly extracts equations in terms of the key, significantly simplifying the attack and drastically reducing the complexity of the linearization attack (if needed)—ultimately breaking the security guarantees of these schemes. Likewise, Hera employs a key-scheduling approach that yields a similar vulnerability.

In contrast, Rubato combines Hera's key-scheduling mechanism with truncation, which should, in principle, strengthen its security. However, the truncation is not leveraged

to its full potential, as it conceals only the last four values regardless of the state size. Furthermore, the scheme primarily relies on added noise for security, which our attack effectively bypasses, leading to a full key recovery using linearization. However, the linearization attack complexity is still large, and we resort to utilizing two differentials to showcase key recovery in real-time for Rubato. This finding suggests that incorporating sufficient truncation and key-whitening techniques can enhance security, as more than a single fault injection would be required to recover the key. Finally, for schemes employing whitening-like techniques, using at least two independent keys may further hinder full key recovery. An interesting direction for future work is to assess the trade-off between the cost of implementing countermeasures against Sasta-DFA and the impact of modifying the scheme design.

7.2 Implication on FHE Privacy Guarantees

The HHE protocol is built on top of FHE and should theoretically maintain the same privacy guarantees. However, in a practical setting, Sasta-DFA demonstrates that full key recovery is possible within the HHE context, resulting in a loss of the privacy guarantees of the underlying FHE schemes as well. Outside the HHE context, specifically with standalone SE schemes, our attack reduces to a standard DFA attack and provides key insights for improving the design of the SE schemes utilized for HHE. This work is the first to show the vulnerability of HHE to DFA while respecting the nonce usage specified by the scheme and practical applications such as SSL/TLS networking protocols. This does not fundamentally compromise the security of FHE itself but highlights the necessity for robust protocol designs and secure implementations when using FHE to protect against physical attacks.

• **Extension of** Sasta-DFA **to** Pasta$_{v2}$ [GLR+24]. In a more recent work [GLR+24], the authors aim to reduce the randomness required by Rasta-like designs, such as Pasta. To achieve this, they propose generating the affine layer of Pasta only once during the first round and reusing it in subsequent rounds. Since the FIP for Pasta in our attack leveraged the Keccak function used to generate the affine layer in the final round, this raises the question: *Does this modification make* Pasta$_{v2}$ *resilient against* Sasta-DFA*?* It is important to note that the applicability of Sasta-DFA in this context depends on implementation choices.

The MDS matrix within the affine layer occupies considerable space and is therefore often generated dynamically when required [ASR25]. If this approach is followed in Pasta$_{v2}$, our attack can still be directly applied, where the fault is induced during the matrix generation step. In this case, the attack range is $\approx 6\%$ similar to the attack range for the Keccak function call, as per the results in [ASR25]. Conversely, if a precomputed matrix is stored and reused, an instruction-skip attack could target the matrix read process (e.g., by skipping the read counter). However, this reduces the available attack range. Thus, while the reduction in randomness [GLR+24] complicates an attacker's efforts and puts the focus on implementation choice, it does not render such attacks impossible and highlights an interesting scope for future research.

8 Future Scope

• **Application and/or Extension.** We induce faults in the last round to showcase the utility of our approach. In the follow-up works [WMT25, WT24], the authors extend this to the last two rounds and also show how our Sasta-DFA model can be applied to the new HHE-enabling schemes (e.g., Elisabeth [CHMS22]) designed for other FHE schemes like TFHE [CGGI20], validating the general applicability of Sasta-DFA to the HHE

setting. Thus, analyzing the standard DFA threat model and applying Sasta-DFA could be valuable, as it aids in removing the nonce-reuse assumption. Furthermore, while we have considered all the schemes over $\mathbb{F}_p$, it is worth exploring HHE schemes designed over $\mathbb{Z}_2$ other than Rasta and AES.

• **Threat Model.** Another area for improvement is the IND-CPA$^{\mathrm{D}}$ threat model [LM21], which allows the attacker to choose the evaluation done on her data. We completely remove this reliance for our attack on AT (Sec. 3.4). Hence, Sasta-DFA on AT does not face function-related constraints and can be used for any privacy-preserving application of FHE, such as private machine learning inference or training. In this case, the attacker does not require prior knowledge of $f()$. However, it would be interesting to see if the same can be extended to the plain setting (Sec. 2.2).

9 Conclusion

This work introduced a novel fault attack technique called Sasta-DFA, designed for mounting DFA-based fault injection attacks on HHE, a protocol based on FHE. Traditional DFA attacks require at least two ciphertext computations with fixed nonce and plaintext assumptions. This is so that fault-free and faulty ciphertexts can be obtained under the same plaintext and public parameters to form an exploitable differential. However, Sasta-DFA eliminates the need for such assumptions in the context of HHE, thus laying the foundation of a realistic and practical attack model for the HHE protocol.

We validated this approach experimentally by executing the first fault injection attack on an off-the-shelf microcontroller running HHE enabling SE firmware for Hera. This proof-of-concept resulted in a unique and full key recovery with a single fault. We further showed how Sasta-DFA can be extended to AT technique built on FHE, eliminating the application dependency. We hope this will inspire the development of robust and secure FHE-based protocol designs in the future for various privacy-preserving applications.

Acknowledgement

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

References

[ADE+23] Ehud Aharoni, Nir Drucker, Gilad Ezov, Eyal Kushnir, Hayim Shaul, and Omri Soceanu. Poster: Efficient AES-GCM decryption under homomorphic encryption. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, pages 3567–3569. ACM, 2023.

[AI] Verrtex AI. Vertex ai. https://cloud.google.com/vertex-ai.

[AM11a] Subidh Ali and Debdeep Mukhopadhyay. A differential fault analysis on AES key schedule using single fault. In Luca Breveglieri, Sylvain Guilley, Israel Koren, David Naccache, and Junko Takahashi, editors, *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011, Tokyo, Japan, September 29, 2011*, pages 35–42. IEEE Computer Society, 2011.

[AM11b] Subidh Ali and Debdeep Mukhopadhyay. An improved differential fault analysis on AES-256. In Abderrahmane Nitaj and David Pointcheval, editors, *Progress in Cryptology - AFRICACRYPT 2011 - 4th International Conference on Cryptology in Africa, Dakar, Senegal, July 5-7, 2011. Proceedings*, volume 6737 of *Lecture Notes in Computer Science*, pages 332–347. Springer, 2011.

[AMK+25] Aikata, Ahmet Can Mert, Sunmin Kwon, Maxim Deryabin, and Sujoy Sinha Roy. REED: Chiplet-Based Scalable Hardware Accelerator for Fully Homomorphic Encryption. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, page 1190, 2025.

[ASR25] Aikata Aikata, Daniel Sanz Sobrino, and Sujoy Sinha Roy. PASTA on edge: Cryptoprocessor for hybrid homomorphic encryption. *IEEE Design and Automation Conference*, page 1919, 2025.

[BBB+10] Alessandro Barenghi, Guido Marco Bertoni, Luca Breveglieri, Mauro Pellicioli, and Gerardo 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*, pages 167–172. IEEE, 2010.

[BBS21] Adda-Akram Bendoukha, Aymen Boudguiga, and Renaud Sirdey. Revisiting stream-cipher-based homomorphic transciphering in the TFHE era. In Esma Aïmeur, Maryline Laurent, Reda Yaich, Benoît Dupont, and Joaquín García-Alfaro, editors, *Foundations and Practice of Security - 14th International Symposium, FPS 2021, Paris, France, December 7-10, 2021, Revised Selected Papers*, volume 13291 of *Lecture Notes in Computer Science*, pages 19–33. Springer, 2021.

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

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

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

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

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

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

[BU02] John Black and Hector Urtubia. Side-channel attacks on symmetric encryption schemes: The case for authenticated encryption. In Dan Boneh, editor, *Proceedings of the 11th USENIX Security Symposium, San Francisco, CA, USA, August 5-9, 2002*, pages 327–338. USENIX, 2002.

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

[CCP+24] Jung Hee Cheon, Hyeongmin Choe, Alain Passelègue, Damien Stehlé, and Elias Suvanto. Attacks against the ind-cpa$^{\mathrm{d}}$ security of exact FHE schemes. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pages 2505–2519. ACM, 2024.

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

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

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

[CHMS22] Orel Cosseron, Clément Hoffmann, Pierrick Méaux, and François-Xavier Standaert. Towards globally optimized hybrid homomorphic encryption - featuring the elisabeth stream cipher. *IACR Cryptol. ePrint Arch.*, page 180, 2022.

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

[CSBB24] Marina Checri, Renaud Sirdey, Aymen Boudguiga, and Jean-Paul Bultel. On the practical cpa$^{\mathrm{d}}$ security of "exact" and threshold FHE schemes and libraries. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part III*, volume 14922 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2024.

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

[DEK+16] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Victor Lomné, and Florian Mendel. Statistical fault attacks on nonce-based authenticated encryption schemes. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 369–395, 2016.

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

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

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

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

[FIPum] 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.

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

[GAH+23] Lorenzo Grassi, Irati Manterola Ayala, Martha Norberg Hovd, Morten Øygarden, Håvard Raddum, and Qingju Wang. Cryptanalysis of symmetric primitives over rings and a key recovery attack on rubato. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part III*, volume 14083 of *Lecture Notes in Computer Science*, pages 305–339. Springer, 2023.

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

[GGF22] Paul A. Grassi, Michael E. Garcia, and James L. Fenton. Digital identity guidelines. *NIST Special Publication 800-63-3*, page 75, 2022.

[GLR+24] Lorenzo Grassi, Fukang Liu, Christian Rechberger, Fabian Schmid, Roman Walch, and Qingju Wang. Minimize the randomness in rasta-like designs: How far can we go? *IACR Cryptol. ePrint Arch.*, page 791, 2024.

[Goo] Google. Medical imaging suite. https://cloud.google.com/medical-imaging.

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

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

[GSR24] Anit Kumar Ghosal, Amit Sardar, and Dipanwita Roychowdhury. Differential fault analysis attack-tolerant hardware implementation of AES. *J. Supercomput.*, 80(4):4648–4681, 2024.

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

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

[HJM+21] Martin Hell, Thomas Johansson, Alexander Maximov, Willi Meier, and Hirotaka Yoshida. Grain-128aeadv2: Strengthening the initialization against key reconstruction. In Mauro Conti, Marc Stevens, and Stephan Krenn, editors, *Cryptology and Network Security - 20th International Conference, CANS 2021, Vienna, Austria, December 13-15, 2021, Proceedings*, volume 13099 of *Lecture Notes in Computer Science*, pages 24–41. Springer, 2021.

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

[HKL+22] Jincheol Ha, Seongkwang Kim, ByeongHak Lee, Jooyoung Lee, and Mincheol Son. Rubato: Noisy ciphers for approximate homomorphic encryption. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory*

and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I, volume 13275 of *Lecture Notes in Computer Science*, pages 581–610. Springer, 2022.

[HMA+24] Haruka Hirata, Daiki Miyahara, Victor Arribas, Yang Li, Noriyuki Miura, Svetla Nikova, and Kazuo Sakiyama. All you need is fault: Zero-value attacks on AES and a new λ-detection m&m. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(1):133–156, 2024.

[HS20] Shai Halevi and Victor Shoup. Design and implementation of HElib: a homomorphic encryption library. Cryptology ePrint Archive, Paper 2020/1481, 2020.

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

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

[LM21] Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. In Anne Canteaut and François-Xavier Standaert, editors, *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*, volume 12696 of *Lecture Notes in Computer Science*, pages 648–677. Springer, 2021.

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

[LNGW21] Brent Lagesse, Gabriel Nguyen, Utsav Goswami, and Kevin Wu. You had to be there: Private video sharing for mobile phones using fully homomorphic encryption. In *19th IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events, PerCom Workshops 2021, Kassel, Germany, March 22-26, 2021*, pages 730–735. IEEE, 2021.

[LSMI21] Fukang Liu, Santanu Sarkar, Willi Meier, and Takanori Isobe. Algebraic attacks on rasta and dasta using low-degree equations. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part I*, volume 13090 of *Lecture Notes in Computer Science*, pages 214–240. Springer, 2021.

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

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

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

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

[Min] MindNetwork. Fhe for web3.

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

[MR24] Pierrick Méaux and Dibyendu Roy. Theoretical differential fault attacks on FLIP and filip. *Cryptogr. Commun.*, 16(4):721–744, 2024.

[Muk09] Debdeep Mukhopadhyay. An improved fault based attack of the advanced encryption standard. In Bart Preneel, editor, *Progress in Cryptology - AFRICACRYPT 2009, Second International Conference on Cryptology in Africa, Gammarth, Tunisia, June 21-25, 2009. Proceedings*, volume 5580 of *Lecture Notes in Computer Science*, pages 421–434. Springer, 2009.

[NLV11] Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In Christian Cachin and Thomas Ristenpart, editors, *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011*, pages 113–124. ACM, 2011.

[NWH+25] Chao Niu, Benqiang Wei, Zhicong Huang, Zhaomin Yang, Cheng Hong, Meiqin Wang, and Tao Wei. SoK: FHE-friendly symmetric ciphers and transciphering. IACR Trans. Cryptogr. Hardw. Embed. Syst., 2025.

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

[PAL21] PALISADE Lattice Cryptography Library (release 1.11.2). https://palisade-crypto.org/, May 2021.

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

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

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

[SKC14] Dhiman Saha, Sukhendu Kuila, and Dipanwita Roy Chowdhury. Escape: Diagonal fault analysis of APE. In Willi Meier and Debdeep Mukhopadhyay, editors, *Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India, New Delhi, India, December 14-17, 2014, Proceedings*, volume 8885 of *Lecture Notes in Computer Science*, pages 197–216. Springer, 2014.

[SMC09] Dhiman Saha, Debdeep Mukhopadhyay, and Dipanwita Roy Chowdhury. A diagonal fault attack on the advanced encryption standard. *IACR Cryptol. ePrint Arch.*, page 581, 2009.

[SMS23] Xhani Marvin Saß, Richard Mitev, and Ahmad-Reza Sadeghi. Oops..! I glitched it again! how to multi-glitch the glitching-protections on ARM trustzone-m. In Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 6239–6256. USENIX Association, 2023.

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

[TBP20] Florian Tramèr, Dan Boneh, and Kenny Paterson. Remote side-channel attacks on anonymous transactions. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 2739–2756. USENIX Association, 2020.

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

[WMT25] Weizhe Wang, Pierrick Méaux, and Deng Tang. Shortcut2secrets: A table-based differential fault attack framework. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2025(2):385–419, 2025.

[WPH+22] Yingchen Wang, Riccardo Paccagnella, Elizabeth Tang He, Hovav Shacham, Christopher W. Fletcher, and David Kohlbrenner. Hertzbleed: Turning power side-channel attacks into remote timing attacks on x86. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 679–697. USENIX Association, 2022.

[WT24] Weizhe Wang and Deng Tang. Differential fault attack on he-friendly stream ciphers: Masta, pasta and elisabeth. *IACR Cryptol. ePrint Arch.*, page 1005, 2024.

Appendix

A The IND-CPA$^{\mathrm{D}}$ Security Overview

The IND-CPA$^{\mathrm{D}}$ (Indistinguishability under Chosen Plaintext Attacks with Decryption oracle) security model was introduced by Li and Micciancio [LM21]. In an FHE scheme $\mathsf{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$, if an adversary knows m, f, and $c_{\text{FHE}} = \mathsf{Enc}(m)$, then access to $\mathsf{Dec}(\mathsf{Eval}(f, c_{\text{FHE}}))$ seems harmless, as they can compute $f(m)$ directly. Since FHE ensures $\mathsf{Dec}(\mathsf{Eval}(f, Enc(m))) = f(m)$ This suggests INS-CPA$^{\mathrm{D}}$ security follows from IND-CPA security, as shown in Figure 12. However, Li and Micciancio [LM21] disproved this for approximate FHE schemes. Their attack exploits the fact that given a ciphertext ($c_{\text{FHE}} = (-a \cdot s + e + m, a)$) after homomorphic encryption of a message m for secret polynomial s and public polynomial a, the decrypted result becomes $m + e$. If an attacker, posing as the client, obtains the result of encryption c_{FHE} and decryption $(m + e)$, she can easily recover the secret s since polynomial a is public. This attack notion was further extended to exact FHE schemes by several follow-up works [CSBB24, CCP+24].

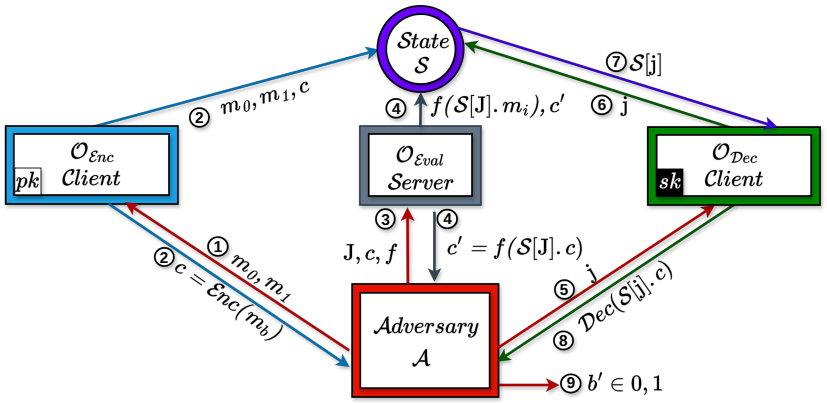

Figure 12: The IND-CPA$^{\mathrm{D}}$ threat model for FHE as introduced in [LM21].

Definition (IND-CPA$^{\mathrm{D}}$ Security Game). Let $\prod$ = (KeyGen, Enc, Dec, Eval) be an FHE scheme with plaintext space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. For an experiment parameterized by a bit $b \in \{0, 1\}$ and an adversary $\mathcal{A}$ that is given access to the following oracles $\mathcal{O}_{Enc}$ (Encryption Oracle), $\mathcal{O}_{Eval}$ (Evaluation Oracle), and $\mathcal{O}_{Dec}$ (Decryption Oracle) sharing a common state $S \in (\mathcal{M} \times \mathcal{M} \times \mathcal{C}^*)$ consisting of message-message-ciphertext triplet sequence:

- Enc: $\mathcal{A}$ submits a pair of plaintext messages (m_0, m_1), to $\mathcal{O}_{Enc}$ which randomly chooses a secret bit b (0 or 1), and computes $c_{\text{FHE}} \leftarrow \mathrm{Enc}_{\mathrm{pk}}(m_b)$ and extends the state $S = [S; (m_0, m_1, c_{\text{FHE}})]$; and returns c_{FHE} to the adversary.

- Eval: $\mathcal{A}$ gives a function $f : \mathcal{M}^k \to \mathcal{M}$ and a sequence of indices $J = (j_1, \cdots, j_k) \in \{1, \cdots, |S|\}^k$ to $\mathcal{O}_{Eval}$ which computes the ciphertext $c_{\text{FHE}} \leftarrow \mathrm{Eval}_{\mathrm{pk}}(f, S[j_1] \cdot c_{\text{FHE}}, \cdots, S[j_k] \cdot c_{\text{FHE}})$, where $S[l_i] \cdot v$ is the element $v \in \{m_0, m_1, c_{\text{FHE}}\}$ of the state S for the l_i-th request. Then, it extends the state with one more triplet $S := [S; (f(S[j_1] \cdot m_0, \cdots, S[j_k] \cdot m_0), f(S[j_1] \cdot m_1, \cdots, S[j_k] \cdot m_1), c_{\text{FHE}})]$. Ciphertext c_{FHE} is returned to the adversary.

- Dec: $\mathcal{A}$ gives an index $j \leq |S|$ to $\mathcal{O}_{Dec}$ who checks whether $S[j] \cdot m_0 = S[j] \cdot m_1$, and, if so, returns $\mathrm{Dec}_{\mathrm{sk}}(S[j] \cdot c_{\text{FHE}})$, otherwise $\perp$ to the adversary.

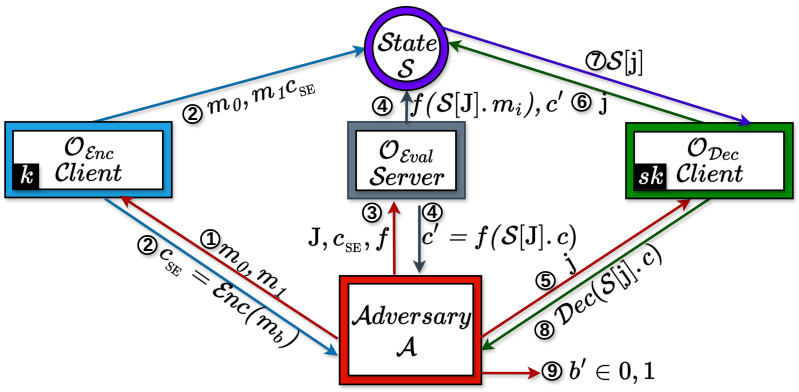

Figure 13: The IND-CPA$^{\mathrm{D}}$ threat model [LM21] for HHE.

- Guessing Stage: After multiple such queries, $\mathcal{A}$ tries to guess b. If correct, $\mathcal{A}$ wins; otherwise, she loses.

It should be noted –

- FHE schemes lack IND-CCA security because a decryption query with a malformed homomorphic ciphertext can reveal the secret key. To avoid this trivial case, assumptions are imposed on the queries. Specifically, the decryption oracle only accepts ciphertexts generated by the encryption or evaluation oracles. In other words, ciphertexts must either result directly from an encryption operation or be obtained through homomorphic evaluation of previously encrypted values. Consequently, only properly formed ciphertexts produced by valid homomorphic operations are permitted. This limitation prevents the adversary from submitting arbitrary ciphertexts for decryption, ensuring that only those created within the game can be queried.
- In this game, the adversary has control over the homomorphic computations performed, as they can specify the function f in the evaluation request.

A.1 IND-CPA$^{\mathrm{D}}$ Security for HHE.

In the context of HHE, $\mathcal{O}_{Enc}$ performs symmetric encryption, and $\mathcal{O}_{Eval}$ first computes homomorphic symmetric decryption and then evaluates f. The use of nonce for the new and efficient stream ciphers (or CBC/Counter mode for AES) ensures every encryption result is distinct, even for the same ciphertext. Thus, the following changes are applied to the above IND-CPA$^{\mathrm{D}}$ security game, as shown in Fig. 13.

- Enc: $\mathcal{A}$ submits a pair of plaintext messages (m_0, m_1), to $\mathcal{O}_{Enc}$ which randomly chooses a secret bit b (0 or 1), and computes $c_{\mathrm{SE}} \leftarrow m_b + \mathtt{E}_\pi(K, n_i)$ and extends the state $S = [S; (m_0, m_1, c_{\mathrm{SE}})]$; and returns c_{SE} to the adversary.
- Eval: $\mathcal{A}$ gives a function $f : \mathcal{M}^k \to \mathcal{M}$ and a sequence of indices $J = (j_1, \cdots, j_k) \in \{1, \cdots, |S|\}^k$ to $\mathcal{O}_{Eval}$ which first computes homomorphic symmetric decryption on c_{SE} to obtain c_{FHE}, and the rest of the steps follow.

A.2 IND-CPA$^{\mathrm{D}}$ Security for HHE in the presence of a Fault.

While prior works [LM21, CSBB24, CCP+24] could attack FHE schemes under the IND-CPA$^{\mathrm{D}}$ security notion, these attacks can be mitigated using noise flooding or rounding [LM21]. Thus, in the setting where the IND-CPA$^{\mathrm{D}}$ attacks are mitigated (or for the FHE schemes [CGGI20] unexploitable under this model), we explore if we can mount a

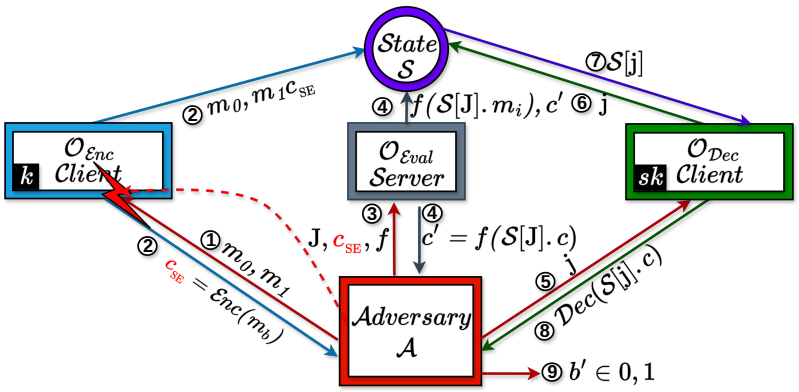

Figure 14: The adaptation of IND-CPA$^{\mathrm{D}}$ threat model [LM21] for HHE to the SASTA model, which extends it by incorporating fault injection capability.

key-recovery attack by inducing fault in the encryption routine in the context of HHE. Notably, we restrict this to a single faulty query. Thus, the changes to the existing definition (Appendix A.1) are as follows (Fig. 14).

- Enc: $\mathcal{A}$ submits a pair of plaintext messages (m_0, m_1), to $\mathcal{O}_{Enc}$ which randomly chooses a secret bit b (0 or 1), and computes $c_{\mathrm{SE}} \leftarrow m_b + \mathtt{E}_\pi(K, n_i)$ during which an adversary induces a fault during computation of $\mathtt{E}_\pi(K, n_i)$, resulting in faulty encryption $c'_{\mathrm{SE}} \leftarrow m_b + \mathtt{E}'_\pi(K, n_i)$.

Note that due to the use of nonce, every $\mathtt{E}_\pi(K, n_i)$ is distinct, and the adversary does not know the current value of $\mathtt{E}_\pi(K, n_i)$. Hence, it can also not predict the faulty value and consequently cannot guess b. The decryption requirement- $S[j] \cdot m_0 = S[j] \cdot m_1$ eliminates trivial attacks where the adversary can distinguish between two computations that lead to different results when computed on exact values. Thus ensuring that the fault induced is non-trivial; for instance, no rounds are skipped, and the addition of $\mathtt{E}_\pi(K, n_i)$ to the message is not bypassed. To further substantiate this in the practical context of the new HHE enabling stream ciphers, we explicitly state that the fault can only be induced in its non-Key-dependent phase (for e.g., the XOF function targeted in this work).

A.3 IND-KPA$^{\mathrm{D}}$ Security for HHE in the presence of a Fault.

The KPA attack allows the adversary to access (plaintext, ciphertext) pairs passively through eavesdropping. Our formal definition of IND-KPA$^{\mathrm{D}}$ (Indistinguishability under Known Plaintext Attacks with Decryption oracle) follows from the IND-CPA$^{\mathrm{D}}$ definition described above. The only difference is that the IND-KPA$^{\mathrm{D}}$ adversary does not have query access to an encryption oracle and cannot request an evaluation on the function of her choice. Thus, this definition augments the IND-KPA security by allowing the adversary to access the encryption and decryption result. Similar to the original setting, the decryption queries are restricted to ciphertexts produced by encryption or evaluation oracles to ensure ciphertext validity and avoid trivial attacks.

B PASTA scheme design overview

In this section, we will give a brief idea of new SE schemes design. For this, we choose PASTA [DGH+23] due to its comprehensive support for various operations. Other designs can be viewed as adaptations or variations of PASTA and are detailed in Appendices C, D,

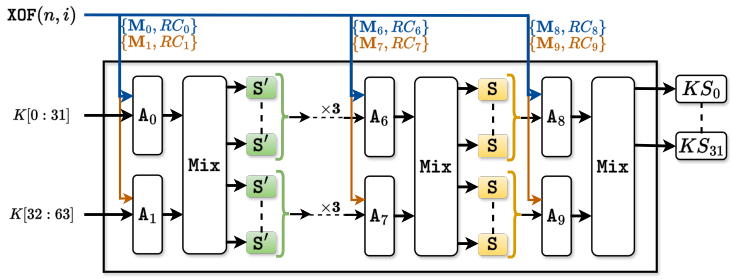

Figure 15: The PASTA-4 encryption takes as input the key (K), nonce (n), and counter (i), and ultimately generates the truncated result- key stream (KS).

and E. In the context of HHE, variables can be categorized into two types: public and private. As illustrated in Fig. 3 and Fig. 15, 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 known exclusively to the client.

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. Fig. 15 demonstrates the PASTA encryption. It is important to note that 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 $\mathbb{F}_p$. Here, p can be any prime between 16 and 60 bits such that $\gcd(p-1, 3) = 1$, 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, $2t = 256$ 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 permutations. The resulting KS is added to the plaintext for encryption and subtracted from the ciphertext for decryption. The round function consists of several layers that are applied in each round, described as follows.

- $\texttt{A}_i$ (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 $\texttt{A}_i$, 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.
- $\texttt{S}'/\texttt{S}$ (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 ($\texttt{S}'$) is utilized. Both these s-boxes are *invertible*.
- Truncation Layer (`Trunc`): This is applied at the end of the PASTA-π permutation and truncates the output to X_L to prevent round inversion.

C MASTA and RASTA design overview

MASTA [HKC+20] follows a slightly different approach as shown in Fig. 16. 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 (Eq. 19) only. 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 $\mathbb{F}_p$, making them suitable for BGV and BFV. Rasta [DEG+18] is similar to Masta with the only difference being operation over $\mathbb{Z}_2$ and the choice of S-box.

$$\mathtt{S}_\chi(X^j) = X^j + X^{j+2} + X^{j+1} \cdot X^{j+2} \pmod{p} \tag{19}$$

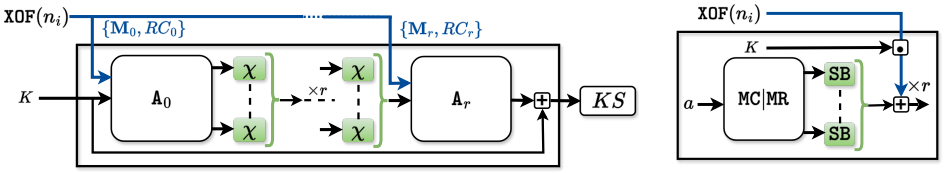

Figure 16: The r-round Masta and a general idea of Hera and Rubato round functions.

D Hera design overview

Hera [CHK+21] adopts a distinct approach compared to Pasta and Masta (as shown in Fig. 16). It comprises five rounds, all utilizing cube S-boxes, thus requiring a modulus such that $\gcd(p-1, 3) = 1$. 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 $\mathbb{F}_p$ (BGV, BFV, and CKKS).

E Rubato design overview

Rubato [HKL+22] was developed after Pasta and Hera, utilizing design principles from both schemes (Fig. 16). 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.

F Keccak and Number Theoretic Transform

The Keccak [PA11] 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) [LN16, Spr] facilitates the conversion of polynomials from coefficient representation to slot representation. With this transformation, polynomial multiplication has a cheap

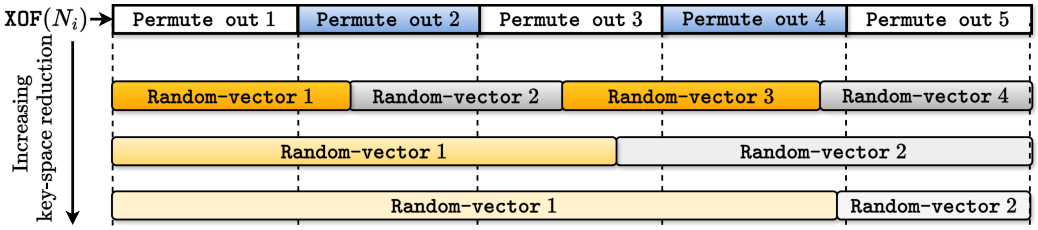

Figure 17: This figure shows that more KECCAK permutations for random vector generation of HERA make it easier to target a specific vector without affecting others, leading to a higher reduction in the keyspace.

$\mathcal{O}(n \log n)$ 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.

G Invertible Matrix Generation

The steps for generating the invertible matrices ($\mathbf{M}_i$) of PASTA-$\mathtt{E}_\pi$ 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 Eq. 20 [GPP11, GPPR11]. This ensures that the matrix is invertible

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

H HERA FIPs study

As stated in Sec. 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 Fig. 17.

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-DFA 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.

I Server-side attack surface

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.

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

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

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 Appendix G. 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. This method encodes the diagonals of matrix $\mathbf{M}$, or the vector in NTT form (discussed in Appendix F). 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 [DRPR19, MDP+20, BFP19, BH22]. 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.

J Comparison Table

Table 7 presents the detailed comparisons with prior works. It shows that while our technique is utilized for HHE, it can also be utilized in standalone symmetric setting with standard assumptions for a key-recovery.

Table 7: More comprehensive comparison of SASTA-DFA with prior works. The average time is reported for key-recovery on a 2GHz clock and 4GB RAM processor. † stands for results reported in [AM11b].

Target Scheme	Security	Field	Nonce Resp.	#Faults	Time	HHE Context	Reference
Flip_{530}	80-bit	$\mathbb{Z}_2$	No	1	39hr	✗	[RBM21]
Kreyvium	128-bit	$\mathbb{Z}_2$		3	5min	✗	[RBM21]
Filip_{430}	80-bit	$\mathbb{Z}_2$		1	751hr	✗	[RKMR23]
RASTA-6	80-bit	$\mathbb{Z}_2$		1	96hr	✗	[RKMR23]
RASTA-5	128-bit	$\mathbb{Z}_2$	**No**	1	0.11s	✗	Section 4.5
RASTA-6	128-bit	$\mathbb{Z}_2$		1	0.12s	✗	Section 4.5
PASTA-3	128-bit	$\mathbb{F}_p$		1	0.50s	✗	Section 4.1
PASTA-4	128-bit	$\mathbb{F}_p$		1	0.21s	✗	Section 4.1
MASTA-4	128-bit	$\mathbb{F}_p$		1	0.09s	✗	Section 4.2
MASTA-5	128-bit	$\mathbb{F}_p$		1	0.07s	✗	Section 4.2
HERA-5	128-bit	$\mathbb{F}_p$		1	0.005s	✗	Section 4.3
RUBATO	128-bit	$\mathbb{F}_p$		2	0.15s	✗	Section 4.4
AES-GCM	128-bit	$\mathbb{Z}_2$	**Yes**	1	5s	✓	Section 4.6
AES-GCM	256-bit	$\mathbb{Z}_2$		1	45min†	✓	Section 4.6
RASTA-5	128-bit	$\mathbb{Z}_2$		1	0.11s	✓	Section 4.5
RASTA-6	128-bit	$\mathbb{Z}_2$		1	0.12s	✓	Section 4.5
PASTA-3	128-bit	$\mathbb{F}_p$		1	0.50s	✓	Section 4.1
PASTA-4	128-bit	$\mathbb{F}_p$		1	0.21s	✓	Section 4.1
MASTA-4	128-bit	$\mathbb{F}_p$		1	0.09s	✓	Section 4.2
MASTA-5	128-bit	$\mathbb{F}_p$		1	0.07s	✓	Section 4.2
HERA-5	128-bit	$\mathbb{F}_p$		1	0.05s	✓	Section 4.3
RUBATO	128-bit	$\mathbb{F}_p$		2	0.15s	✓	Section 4.4