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

Aikata Aikata
Graz University of Technology, Austria
aikata@tugraz.at

Ahaan Dabholkar
Purdue University, United States
adabholk@purdue.edu

Dhiman Saha
Indian Institute of Technology Bhilai, India
dhiman@iitbhilai.ac.in

Sujoy Sinha Roy
Graz University of Technology, Austria
sujoy.sinharoy@tugraz.at

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^{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 the new HHE-enabling schemes using ATXmega128D4-AU microcontroller. Finally, we extend Sasta-DFA to Authenticated Transciphering protocols under a weaker threat model that removes any functional dependency.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ASIACCS'26, Bangalore, India
© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/2026/76
https://doi.org/XXXXXXX.XXXXXXX

CCS Concepts

• **Security and privacy → Cryptography**; **Cryptanalysis and other attacks**; **Public key encryption**; **Symmetric cryptography and hash functions**; **Privacy protections**.

Keywords

DFA, Single Fault, FHE, HHE, Pasta, Masta, Hera, Rubato, Rasta

ACM Reference Format:
Aikata Aikata, Ahaan Dabholkar, Dhiman Saha, and Sujoy Sinha Roy. 2026. SASTA: Single-Fault Nonce Respecting Differential Attack on Hybrid Homomorphic Encryption. In *ACM ASIA Conference on Computer and Communications Security, June 01–06, 2026, Bangalore, India.* ACM, New York, NY, USA, 20 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Fully Homomorphic Encryption (FHE) schemes [12, 13, 19, 20, 30, 33] offer privacy preserving capabilities. However, their widespread adoption is inhibited by significant computational and communication overheads. This is because FHE 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 [3, 32, 59], 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* [61], was introduced. The idea is to replace expensive homomorphic encryption method 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. As a result, this method reduces the client's communication and computation overhead.

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 [27], Masta [42], Hera [21], and Rubato [43], 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.

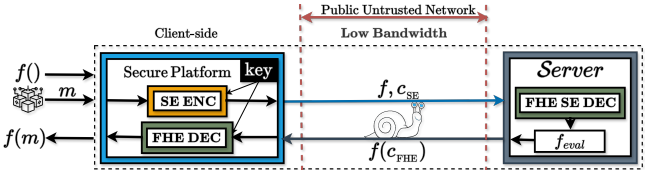

Figure 1: The client symmetrically encrypts the data and sends it to the server, who converts it to FHE ciphertext, and returns the evaluated result. The client then decrypts it.

Recent works [1, 7] have proposed *Authenticated Transciphering* (AT) to further enhance the capabilities of HHE. 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 mode. Its ability to obtain a Tag helps the client to validate the data integrity.

1.1 Research Gap

Physical attacks on cryptographic implementations [8–10, 69, 71, 73] 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 [9], execution time [71], and power consumption [73] or disturbing a cryptographic computation to recover sensitive data [8, 10, 69]. 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 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) [8].

To date, there have been few DFA studies [64, 65] on HHE tailored SE schemes [15, 24, 55, 56]. 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 [72], and require strong assumptions like nonce reuse. However, this deviates from the scheme specifications and is prohibited by cryptographic protocols like SSL/TLS [25, 39]. 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 [15, 24, 55–57]. In contrast, emerging SE algorithms for HHE, such as Pasta [27], Masta [42], Hera [21], and Rubato [43] 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 [62]. 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 Contributions

We present Sasta-DFA, a novel DFA attack on HHE for recovering client secret keys, with three main contributions.

Table 1: Comparison of Sasta-DFA **with prior works. The average time is for key recovery on a 2GHz clock 4GB RAM processor.** † **stands for results from [5] and F for faults.**

Target Scheme	Sec. bits	Field	Nonce Resp.	#F	Time	HHE	Ref.
Flip_{530}	80	$\mathbb{Z}_2$	No	1	39hr	✗	[65]
Kreyvium	128	$\mathbb{Z}_2$		3	5min	✗	[65]
Filip_{430}	80	$\mathbb{Z}_2$		1	751hr	✗	[64]
Rasta-6	80	$\mathbb{Z}_2$		1	96hr	✗	[64]
AES-GCM	128	$\mathbb{Z}_2$	Yes	1	5s	✓	Sec 4.6
AES-GCM	256	$\mathbb{Z}_2$		1	45min†	✓	Sec 4.6
Rasta-5	128	$\mathbb{Z}_2$		1	0.11s	✓	Sec 4.5
Rasta-6	128	$\mathbb{Z}_2$		1	0.12s	✓	Sec 4.5
Pasta-3	128	$\mathbb{F}_p$		1	0.50s	✓	Sec 4.1
Pasta-4	128	$\mathbb{F}_p$		1	0.21s	✓	Sec 4.1
Masta-4	128	$\mathbb{F}_p$		1	0.09s	✓	Sec 4.2
Masta-5	128	$\mathbb{F}_p$		1	0.07s	✓	Sec 4.2
Hera-5	128	$\mathbb{F}_p$		1	0.005s	✓	Sec 4.3
Rubato	128	$\mathbb{F}_p$		2	0.15s	✓	Sec 4.4

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 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) [1] becomes vulnerable to DFA by leveraging the Sasta-DFA 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 [27], Masta [42], Hera [21], Rubato [43]. This further extends to Rasta [24] defined over $\mathbb{Z}_2$. The results 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 exploits 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://github.com/aikata10/SASTA.

• *Countermeasures.* We analyze potential countermeasures for Sasta-DFA. Our analysis shows that while effective countermeasures-such as redundancy/parity-based detection or infective computation-can substantially harden the implementations, they incur latency or storage overheads. This highlights a fundamental trade-off between

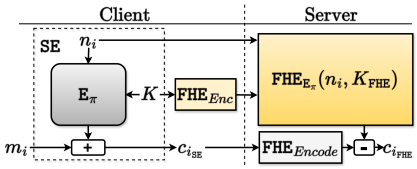

Figure 2: Workflow of HHE: The homomorphically encrypted key (K_{FHE}) is sent once. Next, multiple small ciphertexts ($c_{i_{\mathsf{SE}}}$) encrypted using SE are sent. The server evaluates 'homomorphic SE decryption circuit' (HSD) and obtains a homomorphically encrypted ciphertext $c_{i_{\mathsf{FHE}}}$. Here E_π refers to SE permutation (e.g., AES), and i to i-th iteration of the protocol.

efficiency and physical security, underscoring that protecting HHE against fault attacks might require dedicated design choices over minor implementation fixes.

• *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.

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.

2.1 Hybrid Homomorphic Encryption (HHE)

Following Gentry's first construction [33], several lattice-based FHE schemes have been developed, including BGV [13], BFV [12, 30], CKKS [18, 19], and TFHE [20], typically relying on (R)LWE assumptions. A key limitation of these schemes is ciphertext expansion and the resulting communication and computation overhead, often in the 10^4–$10^5\times$ range [46]. HHE mitigates these costs by combining SE with FHE; we summarize the HHE protocol in Fig. 2 (Table 5).

• The client has two types of keys, the SE 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, allows the attacker to access all messages m_i and results sent by the server.
• Next, the client uses the public key pK to encrypt K using $\mathsf{FHE}_{Encrypt}$ and sends this to the server at the beginning.After this, whenever clients need to store or process data on the cloud, they use SE to encrypt the message blocks m_i using K and send the resultant ciphertext $c_{i_{\mathsf{SE}}} = m_i + \mathsf{E}_\pi$ to the server (nonce n_i is public). E_π refers to the key-dependent permutation that results in a keystream, which is added to plaintext for encryption.
• The server uses the encrypted key $\mathrm{K}_{\mathsf{FHE}}$ along with the nonce n_i to evaluate the SE decryption circuit ($\mathsf{FHE}_{\mathsf{E}_\pi}$) homomorphically (HSD). This results in a ciphertext ($c_{i_{\mathsf{FHE}}}$) that is homomorphically encrypted under key sK and can now be processed.
• The client obtains the resultant ciphertext and decrypts using sK.

AT follows the same framework but replaces SE with an authenticated encryption scheme to provide additional integrity guarantees. A detailed description is given in Appendix A.

2.2 The IND-CPA^{D} Model

IND-CPA (Indistinguishability under Chosen-Plaintext Attack) security ensures that an attacker, even if allowed to encrypt arbitrary messages, cannot distinguish which of two messages was encrypted. While IND-CPA serves as the standard baseline notion of confidentiality, Li and Micciancio [50] argue that it does not fully capture the adversarial capabilities that arise in homomorphic settings, where ciphertexts may be transformed via evaluation and then decrypted. To model this, they introduce the IND-CPA^{D} notion, which extends the IND-CPA experiment by granting the adversary access to a restricted decryption oracle. This oracle is limited to decrypting only valid ciphertexts that result from homomorphic evaluation.

This remains harmless for an FHE scheme $\mathsf{FHE} = (\mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ as long as queries are limited to evaluated ciphertexts [50]. The intuition is straightforward: if an adversary possesses a message m, a function f, and the ciphertext $c_{\mathrm{FHE}} = \mathsf{Enc}(m)$, querying the oracle for the decryption of $\mathsf{Eval}(f, c_{\mathrm{FHE}})$ yields $f(m)$. Since the adversary already knows m and f, they can simply compute $f(m)$ independently. Thus, observing the decryption of the homomorphic computation theoretically provides no additional leverage. However, [50] demonstrate that this intuition fails for approximate FHE schemes like CKKS [19].

Furthermore, subsequent research has shown that even exact schemes, such as BGV [13], can leak "decryption-side information." By forcing the evaluation of carefully chosen functions f, an adversary can cause the error to overflow into the message space, making the secret key exploitable. Countermeasures against such attacks include enforcing strict separation of message and error spaces (rejecting outputs when this separation is violated), applying ciphertext validity checks before releasing decrypted results, and utilizing noise flooding or re-randomization to decorrelate decryption outputs from secret–dependent inputs.

2.3 Differential Fault Analysis of SE

DFA is a physical attack that exploits information leaked from faulty computations on a victim's device. Faults are induced by altering the environmental conditions (e.g., high voltage surges, clock or EM glitches), and the differences between original and faulty outputs are analyzed to reveal secret information. The standard DFA model considers a device that executes encryption/decryption using a fixed secret key, with an adversary that can induce faults and obtain outputs in a chosen-plaintext/ciphertext setting. Most DFA works assume nonce repetition [64, 65]. Since nonce reuse inherently weakens security [26, 67], Dobraunig et al. [25] study a nonce-respecting model for AES-GCM with multiple encryption queries.

Fault attacks on classical SE have been studied extensively, with key-recovery attacks for RSA-CRT signatures [10] and AES [4, 6, 34, 45, 60, 72]. Notably, prior work shows that even a single induced fault during AES can substantially reduce the effective key space, which can be uniquely identified using known plaintext/ciphertext pairs [4, 68, 72]. We therefore focus on DFA rather than SIFA [26] or fault-template attacks, which typically require many faulty and fault-free traces under the same key and, to our knowledge, have not been investigated for newer HHE-enabling schemes. At present, we are not aware of any practically validated fault attack on HHE-enabling SE schemes.

3 The SASTA-DFA Fault Attack Strategy

We adapt the standard DFA framework used in [8, 49, 60, 65, 72] to the HHE domain, proposing a new methodology: SASTA-DFA. Unlike prior DFA works that rely on the strong, often unrealistic assumption of nonce reuse, we leverage the specific structure of the HHE protocol to eliminate this requirement. Furthermore, we restrict the attacker to observing a single query-response pair. This efficiency surpasses existing nonce-respecting fault attacks (e.g., SIFA, FTA), which typically require multiple interactions. To our knowledge, this represents the first systematic exploration of fault attack surfaces at the HHE *protocol* level.

3.1 The Baseline Threat Model

We consider a two-party HHE/AT setting consisting of a Client and an untrusted Server. The Client holds the symmetric key -K used by the HHE-enabled SE layer, as well as the FHE secret key needed to decrypt final evaluated ciphertexts. The Server receives ciphertexts and performs homomorphic evaluation, returning evaluated ciphertexts to the Client.

- Baseline IND-CPA^{D} for FHE. In the IND-CPA^{D} model for FHE, the adversary is allowed restricted access to decryption: it may request decryption only on ciphertexts that arise from homomorphic evaluation (e.g.,Eval(f, c_{FHE})). As detailed in Sec. 2.2, this restriction is typically considered harmless because decryption of a correctly-evaluated ciphertext reveals only the intended function value (e.g., $f(m)$), which is already implied by knowledge of m and f.

- IND-CPA^{D} for HHE. In HHE, the client sends a *symmetric* ciphertext c_{SE} to the server. The server then homomorphically evaluates the SE.Dec circuit, to obtain an FHE ciphertext encoding the corresponding plaintext, after which the computation proceeds as in standard FHE. Thus, although IND-CPA^{D} is an FHE notion, it provides the natural *baseline* guarantee for the FHE back-end used inside HHE, as proved in the following lemma (Appendix B.1).

LEMMA 1 (**IND-CPA^{D}-STYLE SECURITY FOR HHE**). *Let* FHE = (Enc, Dec, Eval) *be an* FHE *scheme and let* SE = (SE.Enc, SE.Dec) *be a symmetric-encryption scheme with key* K. *Consider an* HHE *construction where the client computes* $c_{\mathrm{SE}} \leftarrow \mathsf{SE.Enc}_K(m)$, *the server transforms* c_{SE} *into an* FHE *ciphertext* c_{FHE} , *by homomorphically evaluating* $c_{\mathrm{FHE}} := \mathsf{Eval}(SE.Dec, c_{\mathrm{SE}})$. *It then evaluates the function and outputs* $c^{\star} := \mathsf{Eval}(f, c_{\mathrm{FHE}})$. *The client outputs* $\mathsf{Dec}(c^{\star}) = f(m)$.

Assume the adversary may query Dec *only on* valid evaluated ciphertexts, *that is, on ciphertexts of the form* $\mathsf{Eval}(g, c)$ *for circuits* g *and honestly generated* FHE *ciphertexts* c *arising from the* HHE *pipeline. Then any Probabilistic Polynomial-Time adversary that distinguishes* HHE *encryptions in this experiment can be turned into an adversary that breaks either the required confidentiality of* SE *(e.g., nonce-respecting IND-CPA) or the IND-CPA^{D} security of* FHE.

Hence, if SE *is secure under nonce-respecting IND-CPA and the underlying* FHE *scheme is IND-CPA^{D}-secure, then the resulting* HHE *construction achieves the corresponding IND-CPA^{D}-style confidentiality guarantee in the above restricted-decryption setting.*

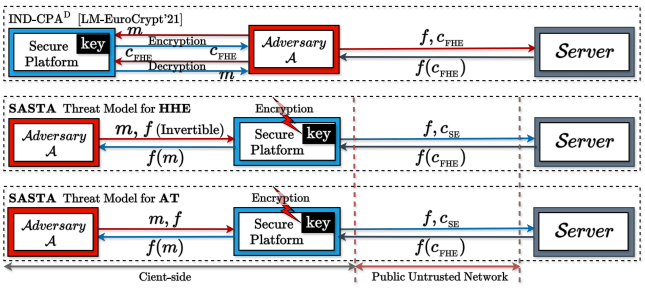

Figure 3: The IND-CPA^{D} threat model [50] and its adaptation to the SASTA-DFA model for HHE and AT in the presence of faults, highlighting adversarial capabilities.

PROOF. The argument proceeds by a sequence of conceptual transformations. First, we modify the experiment by replacing the SE component embedded in the challenge (e.g., the mask/keystream term in stream-/CTR-like instantiations) with uniform randomness. If an adversary can distinguish this modified experiment from the original one, then we can construct an adversary that breaks the confidentiality notion assumed for SE. Next, we further modify the experiment by replacing the FHE ciphertext(s) in the challenge with encryptions of zero, while continuing to answer only the permitted decryption queries, namely decryptions of valid evaluated ciphertexts arising from the HHE pipeline. Any adversary that distinguishes this step can be turned into an adversary that violates the IND-CPA^{D} security of FHE. After these transformations, the adversary's view is independent of the challenge plaintext, and thus its advantage is negligible. □

3.2 The SASTA-DFA Threat Model

The lemma above captures the intuition that breaking confidentiality of the HHE (with restricted decryption of valid evaluated ciphertexts) implies breaking either SE or FHE in IND-CPA^{D}. Our contribution builds on this baseline and studies essentially the same "restricted decryption" setting in the presence of faults during the client-side SE encryption routine, as illustrated in Fig. 3. The resulting ciphertext remains well-formed and proceeds through the standard HHE workflow (security game defined in Appendix B.2).

We therefore study an IND-CPA^{D}+faulty-encryption setting: the decryption interface is not arbitrary, but corresponds to the Client's normal decryption of evaluated ciphertexts returned by the Server; the only additional power is the ability to inject a fault during encryption. This extension is motivated by the intended deployment: encryption is performed on embedded or edge devices, which are well known to be susceptible to physical fault injection (e.g., voltage/clock glitches or EM fault injection), making faults during encryption a realistic assumption in practice.

- Minimal Oracle Requirements. Although we state our security discussion with respect to IND-CPA^{D}, SASTA-DFA does not require its full adversarial capabilities. In IND-CPA^{D}, the adversary may obtain encryptions, receive a challenge ciphertext, and issue restricted decryption queries on valid evaluated ciphertexts (potentially for multiple circuits and inputs). In contrast, our attacker does not query an encryption or decryption oracle in an interactive manner, nor does it require multiple encryption/decryption queries.

It is sufficient that the adversary (i) injects a single fault during client-side symmetric encryption and (ii) observes the plaintext produced upon decryption of the resulting evaluated ciphertext. This ciphertext is well-formed and produced from the standard HHE execution. Notably, since our attack does not rely on the particular decryption-error distribution exploited in [50], it is not prevented by the corresponding "error-distribution" patches; mitigating our attack requires fault-specific countermeasures in addition to existing IND-CPA^{D}-oriented fixes. Details on how SASTA-DFA differs from Standard DFA are provided in Appendix D.

- Why fault extension does not make the setting IND-CCA. The attack model is best viewed as an IND-CPA^{D}-style augmented with faulted encryption. Allowing fault injection during client-side SE encryption does not elevate the adversarial model to IND-CCA (Indistinguishability under Chosen Ciphertext Attack), where the adversary has access to a general decryption oracle that decrypts arbitrary ciphertexts of its choice. SASTA-DFA provides no such access: as in IND-CPA^{D}-style models for HHE, the only plaintexts revealed are those obtained when decrypting valid-evaluated FHE ciphertexts returned by the server.

The fault extension does not change this. Although the adversary may inject a fault during symmetric encryption, it cannot (nor needs to) predict the resulting ciphertext or request decryption of any chosen ciphertext. Under our Fault-Injection Points, the faulty ciphertext remains syntactically well-formed, and its marginal distribution is consistent with honest encryption under the same secret key. Consequently, the adversary does not obtain the arbitrary decryption oracle access assumed in IND-CCA, nor an oracle that outputs ciphertexts from an attacker-controlled distribution.

Moreover, we revisit this point later in Sec. 7, where we explain why verifiable FHE (e.g., via zero-knowledge proofs) used to achieve CCA security at the FHE layer does not mitigate the attack surface captured by our threat model.

- Attack Scope. We study HHE (rather than generic FHE) and focus on the common *masking-based* design, where symmetric encryption derives a mask/keystream from a public nonce and the secret key (as in stream ciphers or CTR mode). This choice is motivated by our fault model, which targets the keystream generation on the encrypting device, and by the fact that all HHE-friendly SE constructions designed for practical use so far follow this stream/CTR pattern. Transciphering in general is not limited to such modes, but the attacks in this work apply to this widely used instantiation.

Finally, our DFA methodology applies to both HHE and AT: for HHE, we assume the evaluated function is non-trivial but invertible, whereas for AT (see Appendix B.3), this assumption can be relaxed and arbitrary functions can be queried (Fig. 3).

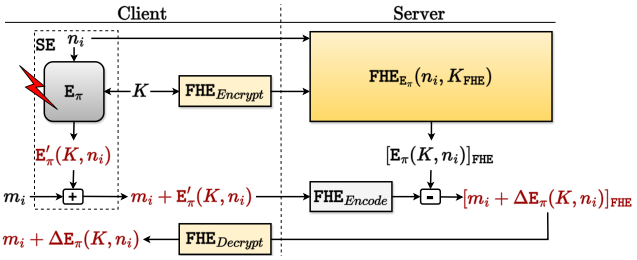

Figure 4: Utilizing SASTA-DFA for differential fault analysis of HHE tailored SE schemes.

- Threat Model in Practical Contexts. SASTA-DFA can be realized in cloud-style settings where decrypted results are publicly available or selectively shared, while keys are protected by a hardware security module or secure enclave and thus remain inaccessible to an attacker on the client. Since encryption is typically performed on embedded/edge devices, often susceptible to physical fault injection (e.g., voltage/clock glitches or EM fault injection), faults during encryption are a realistic assumption. As recent surveys note [62], HHE and AT are increasingly used in practical FHE-based applications; in such deployments, an adversary may fault local symmetric encryption and observe decrypted outputs without directly accessing secret keys.

For instance, this applies to privacy-preserving data sharing and processing used in secure cloud-based medical diagnosis [2, 36], where a hospital encrypts patient data locally and sends it to the cloud for encrypted analysis. A malicious insider who gains access to the device for a short window can see inputs, inject faults, and observe decrypted outputs while keys remain protected by secure hardware. Similar attacks also apply to edge-to-cloud secure video analytics [48], where surveillance systems encrypt video frames locally before uploading them to the cloud for analytics.

3.3 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 a differential (Table 2 ⓑ). This process is further elaborated in Sec. 3.4. 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.

Table 2: Faulty computation flow during HHE. For brevity, we omit the Initialization and Function Evaluation parts.

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

3.4 Attack on HHE

In HHE, the client device simply performs SE encryption, which involves computing key-dependent permutation (E_π) and then adding it to the message, as shown in Fig. 2. 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 2. The variable 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 (E_π) in encryption, the resulting ciphertext c'_i with t coefficients can be expressed as follows.

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

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.

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

Since the original message (m_i) is known, and the attacker can choose any invertible function f, the differential ($\Delta\mathsf{E}_\pi$) can be obtained by simply subtracting the known message m_i from m'_i, as shown in Table 2 ⓒ. Thus, when a fault occurs in the round function computation (E_π) 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.

In Appendix C, we extend our analysis to AT for completeness, and show that the differential derived for HHE is exactly the one needed for AT key recovery. Consequently, the same key-recovery techniques apply without modification.

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. 3.4 or Appendix. C) for key-recovery.

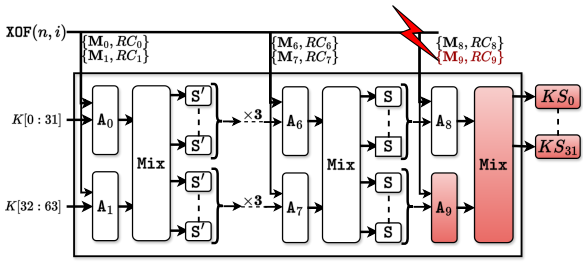

Figure 5: Attack strategy and FIP for Pasta-4 encryption, where $KS = \mathsf{E}_\pi(K, n_i)$.

4.1 Finding the Key to Pasta

The Pasta-4 round function is shown in Fig. 5 (detailed in Appendix E). It consists of four rounds each comprising an Affine Layer A(), a Mixing layer Mix(), and S-box S′/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 ($KS = \mathsf{E}_\pi(K, n_i)$).

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 S operation will result in its diffusion across the entire state due to the subsequent matrix multiplications ($\mathsf{A}_{8/9}$) and 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 A_9, as depicted in Fig. 5. If we denote the input state after the last S operation as $X_8||X_9$, the fault-free result is as follows.

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

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

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

If $\Delta\mathbf{M}_9$ is known and invertible, X_9 and consequently X_8 can be obtained via Eq. 4 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 [47], next we discuss how we choose appropriate fault injection points.

Locating Interesting FIPs (the Achilles Heel). 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 [23, 47, 66] 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 I) *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 for simplicity.

4.2 Extending the Attack to Masta

The Masta [42] scheme was introduced concurrently with Pasta and can be viewed as a direct adaptation of Rasta [24], originally defined over $\mathbb{Z}_2$, to $\mathbb{F}_p$. As outlined in Appendix F, 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 that ensures it is invertible by design (Appendix J). 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.

$$\mathsf{E}_\pi(K, n_i) = \mathtt{A}_r(X_r) + K \pmod{p} \tag{5}$$

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

The fault is induced in the matrix $\mathbf{M}_r$ (Fig. 16 Appendix F), and the resulting differential is as presented below.

$$\Delta\mathsf{E}_\pi = \mathbf{M}_r \cdot X_r - \mathbf{M}'_r \cdot X_r \pmod{p} \tag{7}$$

$$\Delta\mathsf{E}_\pi = \Delta\mathbf{M}_r \cdot X_r \pmod{p} \tag{8}$$

The only unknowns in the above equations are X_r and K. Given that the difference matrix is known and invertible, Eq. 8 can be used to retrieve X_r. No truncation is done here. Instead, the key is added in the end, as shown in Eq. 6. 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 [27]. 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 [27] as follows:

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

The linearization attack operates with a time complexity of $O(b^\omega)$, a memory complexity of $O(b^2)$, and a data complexity of $O(b)$. Here, ω is the linear algebra constant, and while typically $2 < \omega \leq 3$, the more conservative and realistic range is $2.37 \leq \omega \leq 2.82$ [37, 51]. 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. 6 the total number of unknown key variables also reduce to $n_v - rk$, and the total monomials for linearization reduce as follows.

$$b_{\text{SASTA}} = \sum_{i=1}^{d} \binom{n_v - rk + i - 1}{i} \tag{9}$$

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 which is significantly below the 128-bit security threshold, as shown in Table 3. 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. This is detailed in Table 3, which analyzes the security of the HHE-enabling SE schemes before and after the Sasta-DFA attack. It shows the security bounds in the context of the linearization attack $\log_2 b^\omega$, which is greater than 128, as the scheme parameters support a 128-bit security level.

After Sasta-DFA, the required effort to recover the key drops sharply for all schemes $\log_2 b^\omega_{\text{SASTA}}$. For example, in Masta and Rasta (Sec. 4.5), if the faulty matrix is not fully invertible ($n_v - 1 \leq rank(rk) \leq n_v - 4$), the number of unknowns becomes smaller, making key recovery much easier ($\approx 2^{10} - 2^{14}$). In Hera (Sec. 4.3) and Rubato (Sec. 4.4), even when the fault spreads only partly through the state, the attack still creates new simple equations that help recover the key in practical time ($\approx 2^{19} - 2^{38}$). The results for Rubato, which relies on added noise for security, show that Sasta-DFA reduces the security to much lower levels. Overall, Table 3 clearly shows that a single fault can weaken the protection of these designs far below the expected 128-bit security level, even when the resulting differential results in non-invertible matrices or have limited fault diffusion.

4.3 Extending the Attack to Hera

Hera [21], elaborated in Appendix G, takes a slightly different design approach and uses a pseudo-key-schedule, where the XOF output vector (RV) is multiplied with the key and then added to the state (X). This is followed by the affine layer, which has constant

Table 3: Security bounds on the design parameters required to achieve 128-bit security, both before and after SASTA-DFA, in the context of linearization attacks. In addition, we address scenarios where differential matrices are non-invertible (MASTA, RASTA) or key recovery is incomplete due to limited fault diffusion (HERA) or truncation (RUBATO).

Scheme	n_v	d	$\log_2 b$	$\log_2 b^\omega$		$n_v - rk$	$\log_2 b$ SASTA	$\log_2 b^\omega_{\text{SASTA}}$	
				$\omega = 2.37$	$\omega = 2.82$			$\omega = 2.37$	$\omega = 2.82$
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.

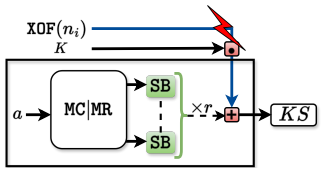

Figure 6: The fault injection points for inducing a fault in HERA or RUBATO permutation before final add-round-key.

matrices, unlike PASTA and MASTA. A fault in the last XOF call utilizing RV_r for state X_r (Fig. 6) would result in the following:

$$\mathsf{E}_\pi(K, n_i) = X_r + RV_r \cdot K \pmod p \quad (10)$$

$$\Delta\mathsf{E}_\pi = RV_r' \cdot K - RV_r \cdot K \pmod p = \Delta RV_r \cdot K \pmod p \quad (11)$$

In the case of HERA, the known difference is simply a vector ΔRV_r, hence recovering key K, using Eq. 11, 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 K 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 Table 3, when additional equations are obtained from SASTA-DFA (Eq. 4.2), 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 [43] (Appendix H) 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. 10 as follows.

$$\mathsf{E}_\pi(K, n_i) = X_r + RV_r \cdot K + e \pmod p \quad (12)$$

After fault injection, the differential is expressed as follows.

$$\Delta\mathsf{E}_\pi = \Delta RV_r \cdot K + e \pmod p \quad (13)$$

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 [19]. 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 [43] as follows.

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

The authors in [37] 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. 13. These equations effectively reduce the influence of the added Gaussian noise, thereby decreasing the number of monomials as follows.

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

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 Table 3. However, after applying Sasta-DFA, this complexity reduces to $2^{115}/2^{77}$ (or $2^{98}/2^{65}$ for $\omega = 2.37$), eliminating any reliance on the added Gaussian noise for security. While this analysis demonstrates that the security assumptions are significantly weakened by Sasta-DFA, executing this linearization attack in real time is challenging.

Therefore, for Rubato, we investigate whether utilizing two differentials could enable full key recovery. We do not impose any additional requirements on the two differentials. We obtain two sets of Eq.12,13, where, as expected for two distinct encryptions, 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. With this second independent faulty query, we successfully recovered the correct key 99.95% of the time, as confirmed over 100,000 test 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 $2^{21}/2^{38}$ (Table 3). Hence, Rubato remains exploitable 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 [24], that utilize XOF in its construction. The only dependency is that there are fewer invertible matrices in $\mathbb{Z}_2$ [47] (≈ 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 [47]. 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 (Table 3). It is important to note that the key in this case is boolean, so an exhaustive search would have a complexity of 2^4, making it a more efficient attack strategy for Rasta. We obtain the differential as follows.

$$\mathsf{E}_\pi(K, n_i) = \mathsf{A}_r(X_r) \oplus K \qquad = \mathbf{M}_r \cdot X_r \oplus RC_r \oplus K \qquad (14)$$

$$\Delta\mathsf{E}_\pi = \mathbf{M}_r \cdot X_r \oplus \mathbf{M}'_r \cdot X_r \qquad = \Delta\mathbf{M}_r \cdot X_r \qquad (15)$$

As discussed in Appendix F, Rasta has similar design principles as Masta [42]. The implementation of Rasta also uses Keccak for XOF, thus making it the 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, a fault in Keccak will also affect RC_r, whose 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 [4–6, 34, 45, 60, 68, 72]. These works also include both single or multiple-fault-based key-recovery techniques [4, 72], [34, 45, 60]. 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 DFA methodology 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 L. 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.

5 The Ambush: Experimental Fault Attack

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[1]. 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 [69], we also utilize the open-source ChipWhisperer toolchain[2]. 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[3]. 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. 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 [11, 14, 28, 58, 69] 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.

To successfully demonstrate fault injection on the Keccak permutation, we use the FIPS202 standard implementation [31]. Our target platform is CWLITEXMEGA, and we employ clock-glitching to induce faults. The design runs as firmware at the default clock

[1] https://rtfm.newae.com/Targets/CW303%20XMEGA/
[2] https://github.com/newaetech/chipwhisperer
[3] The ChipWhisperer toolchain runs AES firmware and supports AES-GCM mode.

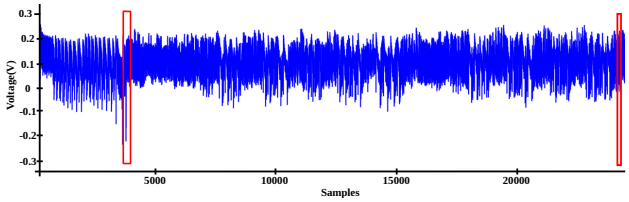

Figure 7: Sample fault injection targets in our work include the start of the KECCAK permutation and the point where its output is written to the final buffer.

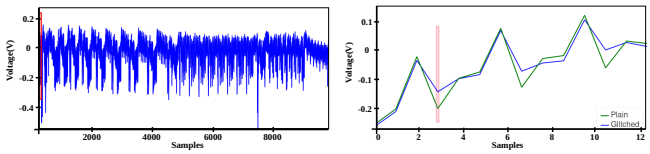

Figure 8: The first plot shows fault injection during the final HERA ARK operation. The second provides a zoomed view of normal and faulty outputs after the clock glitch. Traces are extracted from the original plot, with a relative x-axis.

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. 7 illustrates the power trace and the two specific locations where we injected our attack. The first location corresponds to the start of the KECCAK Permute operation. Injecting a fault here resulted in skipping this particular instruction, causing the output from the preceding permutation to be 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. Similarly, voltage glitches [69] can also be utilized to mount precise attacks during storage of KECCAK intermediate states.

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. 8 displays the power trace for the final Add-round-key operation. We aim for the entire state to be faulty, and 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.

Note that our attack does not necessitate power traces shown in Fig. 8. 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

Our attack requires knowledge of the specific operation affected. Numerous works [11, 14, 28, 58, 69] have shown this to be possible with high precision. We also validated this through our fault attack above. However, in cases where attackers lack precise fault injection, an alternative approach can be employed. Attacker induces an unknown fault and relies on the fault injection range within which the fault is triggered. Since the firmware is known, the attacker simulates all faults in an XOF within the attack range (template), recovering potential faulty outputs and corresponding key guesses. Finally, the attacker recovers the key with a single fault-free result via post-encryption equality check and elimination.

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. 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. An extension to earlier rounds would further increase this range. We show our attack in the last round. This implies countermeasures are only required to protect the last round. Extending this attack to earlier rounds implies more protection, 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 the average is four.

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.

Finally, while our current proof-of-concept targets a microcontroller platform, the same Sasta-DFA principles can extend to FPGA or ASIC implementations. However, we note that clock-glitch effects in parallel modules may increase simultaneous fault propagation, potentially reducing the differential's predictability. Therefore, localized power or EM-based faults can be more effective in targeting specific logic regions, as shown in related FPGA studies [54].

6 Countermeasure analysis

i. 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.

ii. Redundancy-based fault detection. In the long run, redundant computations would be a more suitable countermeasure. 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.

iii. Parity-based fault detection. Since Sasta-DFA targets faults during the SHAKE/Keccak generation of the affine layer, a natural defense is to protect the XOF itself using concurrent error detection (CED). Prior works propose lightweight parity-based checks tailored to Keccak's round operations, for instance, maintaining compact parity information over the 5×5 lane state [53], and more recent designs exploit the structured Keccak "cube" to deploy two-dimensional parity variants [29]. Such error-detecting codes can detect a broad class of transient/random faults with lower overhead than full time/space duplication. In our case, we use instruction skips, and skips inside the Keccak function will break parity propagation, thus highly likely to get detected. Keccak-level CED should be paired with hardened control flow and a fail-closed response (e.g., infective handling) once a mismatch is observed.

iv. Infective Countermeasure. Since a stronger adversary can bypass fault detection, infective countermeasure [35] 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. On the contrary, if there is a fault, the values XORed back to the state would introduce random garbage.

v. Algorithmic Countermeasures. To protect AT, it is best to ensure that the tag sent by the server cannot be utilized to form the differential required for Sasta-DFA (Appendix. C). 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 as discussed in [1]. This incurs a computation overhead on the server side.

vi. Effect of reducing randomness (e.g., Pasta$_{v2}$ [38]). We emphasize that changes that reduce per-round randomness can also change (often shrink) the available fault surface. For example, in a more recent work [38], 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*, thus making it an effective countermeasure?* It is important to note that this does not eliminate the threat: if the matrix is still generated dynamically due to memory constraints of the embedded devices, the same fault strategy can apply at generation time. On the contrary, if the matrix is generated once, stored, and reused, faulting the read/counter logic becomes the alternative, typically with a reduced attack range. Overall, lowering randomness can harden implementations in practice, but only when coupled with careful generation/storage choices and integrity checks.

vii. Design-level considerations. 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, 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, enabling full key recovery via 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 multiple fault injections would be required to recover the key. Yet, improving robustness against Sasta-DFA requires fault-aware design choices rather than tweaks aimed at preventing inversion.

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.

6.1 Countermeasure Performance Evaluation

We report performance evaluation results based on the reference implementation provided by the authors of Pasta [27]. The baseline implementation consumes 274, 203 clock cycles on average

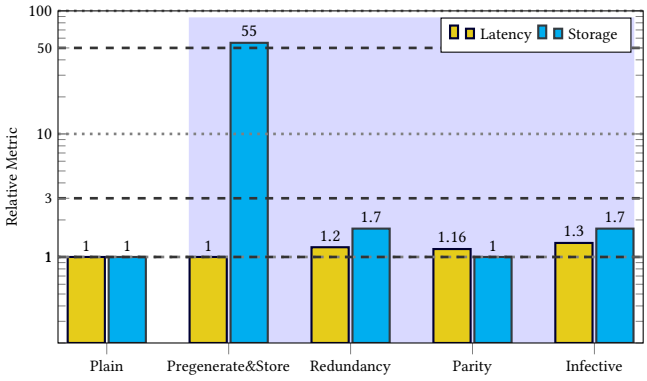

Figure 9: A comparison of latency and storage for different Pasta-**4 implementations (with or without countermeasures). The four highlighted implementations are effective against** Sasta-DFA**. The graph is on a logarithmic scale.**

for 100,000 executions of Pasta-E_π. The implementation results are reported for the 12th Gen Intel i7 CPU. In Fig. 9, we estimate the cost of countermeasures regarding latency and required extra storage. For storage requirements, we estimate the cost in terms of maximum intermediate storage required at any point in time during the Pasta-E_π execution. Fig. 9 shows that parity-based Keccak/XOF protection incurs only limited additional storage, as it maintains compact check information, and avoids the substantial memory overhead of pre-generation and storage. In terms of latency, it is significantly cheaper than duplicating the full XOF computation. Nevertheless, as this protection applies only to the Keccak-based generation stage, it should be combined with countermeasures for the remaining rounds (e.g., infective) to provide an end-to-end hardened implementation.

7 Discussion and Future Scope

• **Impact of CCA-secure or verifiable FHE.** A natural question is whether strengthening the FHE back end, for example, by adopting a CCA-secure or verifiable FHE instantiation (via zero-knowledge proofs), would preclude our attack. It does not. CCA-style security and verifiable-FHE mechanisms are designed to detect or prevent active misuse at the FHE layer, such as submitting malformed FHE ciphertexts, or deviating from the intended evaluation. By contrast, Sasta-DFA injects a fault before the FHE layer, during client-side SE encryption. The resulting ciphertext remains syntactically well-formed and is processed by the honest HHE pipeline, yielding a valid evaluated FHE ciphertext whose decryption is also well-defined. Therefore, even if the server proves correct evaluation (or the client proves ciphertext well-formedness / correct decryption) at the FHE layer, these checks do not eliminate the fault-induced differential introduced at the SE layer, and hence do not prevent Sasta-DFA.

• **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.

• **Application and/or Extension.** We induce faults in the last round to showcase the utility of our approach. In several peer-reviewed follow-up works, 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 [22]) designed for other FHE schemes like TFHE [20], 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^D threat model [50], which allows the attacker to choose the evaluation done on her data. We completely remove this reliance for our attack on AT (Appendix. C). 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. 2.1).

8 Conclusion

This work introduced a novel fault attack technique, called Sasta-DFA, designed to mount 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 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 full key recovery with a single fault. We further showed how Sasta-DFA can be extended to the 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

[1] Ehud Aharoni, Nir Drucker, Gilad Ezov, Eyal Kushnir, Hayim Shaul, and Omri Soceanu. 2023. Poster: Efficient AES-GCM Decryption Under Homomorphic Encryption. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda (Eds.). ACM, 3567–3569. doi:10.1145/3576915.3624377

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

[3] Aikata, Ahmet Can Mert, Sunmin Kwon, Maxim Deryabin, and Sujoy Sinha Roy. 2025. REED: Chiplet-Based Scalable Hardware Accelerator for Fully Homomorphic Encryption. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* (2025), 1190. https://eprint.iacr.org/2023/1190

[4] Subidh Ali and Debdeep Mukhopadhyay. 2011. A Differential Fault Analysis on AES Key Schedule Using Single Fault. In *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011, Tokyo, Japan, September 29, 2011*, Luca Breveglieri, Sylvain Guilley, Israel Koren, David Naccache, and Junko Takahashi (Eds.). IEEE Computer Society, 35–42. doi:10.1109/FDTC.2011.10

[5] Subidh Ali and Debdeep Mukhopadhyay. 2011. An Improved Differential Fault Analysis on AES-256. In *Progress in Cryptology - AFRICACRYPT 2011 - 4th International Conference on Cryptology in Africa, Dakar, Senegal, July 5-7, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6737)*, Abderrahmane Nitaj and David Pointcheval (Eds.). Springer, 332–347. doi:10.1007/978-3-642-21969-6_21

[6] Alessandro Barenghi, Guido Marco Bertoni, Luca Breveglieri, Mauro Pellicioli, and Gerardo Pelosi. 2010. Fault attack on AES with single-bit induced faults. In *Sixth International Conference on Information Assurance and Security, IAS 2010, Atlanta, GA, USA, August 23-25, 2010*. IEEE, 167–172. doi:10.1109/ISIAS.2010.5604061

[7] Adda-Akram Bendoukha, Aymen Boudguiga, and Renaud Sirdey. 2021. Revisiting Stream-Cipher-Based Homomorphic Transciphering in the TFHE Era. In *Foundations and Practice of Security - 14th International Symposium, FPS 2021, Paris, France, December 7-10, 2021, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 13291)*, Esma Aïmeur, Maryline Laurent, Reda Yaich, Benoît Dupont, and Joaquín García-Alfaro (Eds.). Springer, 19–33. doi:10.1007/978-3-031-08147-7_2

[8] Eli Biham and Adi Shamir. 1997. Differential Fault Analysis of Secret Key Cryptosystems. In *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings (Lecture Notes in Computer Science, Vol. 1294)*, Burton S. Kaliski Jr. (Ed.). Springer, 513–525. doi:10.1007/BFb0052259

[9] John Black and Hector Urtubia. 2002. Side-Channel Attacks on Symmetric Encryption Schemes: The Case for Authenticated Encryption. In *Proceedings of the 11th USENIX Security Symposium, San Francisco, CA, USA, August 5-9, 2002*, Dan Boneh (Ed.). USENIX, 327–338. http://www.usenix.org/publications/library/proceedings/sec02/black.html

[10] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. 1997. On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract). In *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding (Lecture Notes in Computer Science, Vol. 1233)*, Walter Fumy (Ed.). Springer, 37–51. doi:10.1007/3-540-69053-0_4

[11] Claudio Bozzato, Riccardo Focardi, and Francesco Palmarini. 2019. Shaping the Glitch: Optimizing Voltage Fault Injection Attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019, 2 (2019), 199–224. doi:10.13154/tches.v2019.i2.199-224

[12] Zvika Brakerski. 2012. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings (Lecture Notes in Computer Science, Vol. 7417)*, Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer, 868–886. doi:10.1007/978-3-642-32009-5_50

[13] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2011. Fully Homomorphic Encryption without Bootstrapping. *Electron. Colloquium Comput. Complex.* (2011), 111. https://eccc.weizmann.ac.il/report/2011/111

[14] Jakub Breier and Xiaolu Hou. 2022. How Practical Are Fault Injection Attacks, Really? *IEEE Access* 10 (2022), 113122–113130. doi:10.1109/ACCESS.2022.3217212

[15] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. 2018. Stream Ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression. *J. Cryptol.* 31, 3 (2018), 885–916. doi:10.1007/s00145-017-9273-9

[16] Marina Checri, Renaud Sirdey, Aymen Boudguiga, and Jean-Paul Bultel. 2024. On the Practical CPA^{D} Security of "exact" and Threshold FHE Schemes and Libraries. In *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 14922)*, Leonid Reyzin and Douglas Stebila (Eds.). Springer, 3–33. doi:10.1007/978-3-031-68382-4_1

[17] Jung Hee Cheon, Hyeongmin Choe, Alain Passelègue, Damien Stehlé, and Elias Suvanto. 2024. Attacks Against the IND-CPA^{D} Security of Exact FHE Schemes. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM, 2505–2519. doi:10.1145/3658644.3690341

[18] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. A Full RNS Variant of Approximate Homomorphic Encryption. In *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 11349)*, Carlos Cid and Michael J. Jacobson Jr. (Eds.). Springer, 347–368. doi:10.1007/978-3-030-10970-7_16

[19] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10624)*, Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer, 409–437. doi:10.1007/978-3-319-70694-8_15

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

[21] Jihoon Cho, Jincheol Ha, Seongkwang Kim, ByeongHak Lee, Joohee Lee, Jooyoung Lee, Dukjae Moon, and Hyojin Yoon. 2021. Transciphering Framework for Approximate Homomorphic Encryption. In *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 13092)*, Mehdi Tibouchi and Huaxiong Wang (Eds.). Springer, 640–669. doi:10.1007/978-3-030-92078-4_22

[22] Orel Cosseron, Clément Hoffmann, Pierrick Méaux, and François-Xavier Standaert. 2022. Towards Globally Optimized Hybrid Homomorphic Encryption - Featuring the Elisabeth Stream Cipher. *IACR Cryptol. ePrint Arch.* (2022), 180. https://eprint.iacr.org/2022/180

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

[24] Christoph Dobraunig, Maria Eichlseder, Lorenzo Grassi, Virginie Lallemand, Gregor Leander, Eik List, Florian Mendel, and Christian Rechberger. 2018. Rasta: A Cipher with Low ANDdepth and Few ANDs per Bit. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10991)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, 662–692. doi:10.1007/978-3-319-96884-1_22

[25] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Victor Lomné, and Florian Mendel. 2016. Statistical Fault Attacks on Nonce-Based Authenticated Encryption Schemes. In *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 (Lecture Notes in Computer Science, Vol. 10031)*, Jung Hee Cheon and Tsuyoshi Takagi (Eds.). 369–395. doi:10.1007/978-3-662-53887-6_14

[26] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. 2018. SIFA: Exploiting Ineffective Fault Inductions on Symmetric Cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018, 3 (2018), 547–572. doi:10.13154/tches.v2018.i3.547-572

[27] Christoph Dobraunig, Lorenzo Grassi, Lukas Helminger, Christian Rechberger, Markus Schofnegger, and Roman Walch. 2023. Pasta: A Case for Hybrid Homomorphic Encryption. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2023, 3 (2023), 30–73. doi:10.46586/tches.v2023.i3.30-73

[28] Jean-Max Dutertre, Timothée Riom, Olivier Potin, and Jean-Baptiste Rigaud. 2019. Experimental Analysis of the Laser-Induced Instruction Skip Fault Model. In *Secure IT Systems - 24th Nordic Conference, NordSec 2019, Aalborg, Denmark, November 18-20, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11875)*, Aslan Askarov, René Rydhof Hansen, and Willard Rafnsson (Eds.). Springer, 221–237. doi:10.1007/978-3-030-35055-0_14

[29] Christian Ewert, Amrit Sharma Poudel, Mouadh Ayache, Andrija Neskovic, Rainer Buchty, Mladen Berekovic, Sebastian Berndt, and Saleh Mulhem. 2025. Lightweight Unified Sha-3/Shake Architecture with a Fault-Resilient State. arXiv:2512.03616 [cs.AR]. doi:10.48550/arXiv.2512.03616

[30] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptol. ePrint Arch.* (2012), 144. http://eprint.iacr.org/2012/144

[31] FIPS202 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.

[32] Robin Geelen, Michiel Van Beirendonck, Hilder V. L. Pereira, Brian Huffman, Tynan McAuley, Ben Selfridge, Daniel Wagner, Georgios Dimou, Ingrid Verbauwhede, Frederik Vercauteren, and David W. Archer. 2022. BASALISC: Flexible Asynchronous Hardware Accelerator for Fully Homomorphic Encryption. doi:10.48550/ARXIV.2205.14017

[33] Craig Gentry. 2009. *A fully homomorphic encryption scheme*. Ph. D. Dissertation. Stanford University, USA. https://searchworks.stanford.edu/view/8493082

[34] Anit Kumar Ghosal, Amit Sardar, and Dipanwita Roychowdhury. 2024. Differential fault analysis attack-tolerant hardware implementation of AES. *J. Supercomput.* 80, 4 (2024), 4648–4681. doi:10.1007/S11227-023-05632-2

[35] Shamit Ghosh, Dhiman Saha, Abhrajit Sengupta, and Dipanwita Roy Chowdhury. 2015. Preventing Fault Attacks Using Fault Randomization with a Case Study on AES. In *Information Security and Privacy - 20th Australasian Conference, ACISP 2015, Brisbane, QLD, Australia, June 29 - July 1, 2015, Proceedings (Lecture Notes in Computer Science, Vol. 9144)*, Ernest Foo and Douglas Stebila (Eds.). Springer, 343–355. doi:10.1007/978-3-319-19962-7_20

[36] Google. [n. d.]. Medical Imaging Suite. https://cloud.google.com/medical-imaging.

[37] Lorenzo Grassi, Irati Manterola Ayala, Martha Norberg Hovd, Morten Øygarden, Håvard Raddum, and Qingju Wang. 2023. Cryptanalysis of Symmetric Primitives over Rings and a Key Recovery Attack on Rubato. In *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 14083)*, Helena Handschuh and Anna Lysyanskaya (Eds.). Springer, 305–339. doi:10.1007/978-3-031-38548-3_11

[38] Lorenzo Grassi, Fukang Liu, Christian Rechberger, Fabian Schmid, Roman Walch, and Qingju Wang. 2024. Minimize the Randomness in Rasta-Like Designs: How Far Can We Go? *IACR Cryptol. ePrint Arch.* (2024), 791. https://eprint.iacr.org/2024/791

[39] Paul A. Grassi, Michael E. Garcia, and James L. Fenton. 2022. Digital Identity Guidelines. *NIST Special Publication 800-63-3* (2022), 75. doi:10.6028/NIST.SP.800-63-3

[40] Jian Guo, Thomas Peyrin, and Axel Poschmann. 2011. The PHOTON Family of Lightweight Hash Functions. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6841)*, Phillip Rogaway (Ed.). Springer, 222–239. doi:10.1007/978-3-642-22792-9_13

[41] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. 2011. The LED Block Cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6917)*, Bart Preneel and Tsuyoshi Takagi (Eds.). Springer, 326–341. doi:10.1007/978-3-642-23951-9_22

[42] Jincheol Ha, Seongkwang Kim, Wonseok Choi, Jooyoung Lee, Dukjae Moon, Hyojin Yoon, and Jihoon Cho. 2020. Masta: An HE-Friendly Cipher Using Modular Arithmetic. *IEEE Access* 8 (2020), 194741–194751. doi:10.1109/ACCESS.2020.3033564

[43] Jincheol Ha, Seongkwang Kim, ByeongHak Lee, Jooyoung Lee, and Mincheol Son. 2022. Rubato: Noisy Ciphers for Approximate Homomorphic Encryption. In *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 13275)*, Orr Dunkelman and Stefan Dziembowski (Eds.). Springer, 581–610. doi:10.1007/978-3-031-06944-4_20

[44] Martin Hell, Thomas Johansson, Alexander Maximov, Willi Meier, and Hirotaka Yoshida. 2021. Grain-128AEADv2: Strengthening the Initialization Against Key Reconstruction. In *Cryptology and Network Security - 20th International Conference, CANS 2021, Vienna, Austria, December 13-15, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 13099)*, Mauro Conti, Marc Stevens, and Stephan Krenn (Eds.). Springer, 24–41. doi:10.1007/978-3-030-92548-2_2

[45] Haruka Hirata, Daiki Miyahara, Victor Arribas, Yang Li, Noriyuki Miura, Svetla Nikova, and Kazuo Sakiyama. 2024. 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 (2024), 133–156. doi:10.46586/TCHES.V2024.I1.133-156

[46] Wonkyung Jung, Eojin Lee, Sangpyo Kim, Jongmin Kim, Namhoon Kim, Keewoo Lee, Chohong Min, Jung Hee Cheon, and Jung Ho Ahn. 2021. Accelerating Fully Homomorphic Encryption Through Architecture-Centric Analysis and Optimization. *IEEE Access* 9 (2021), 98772–98789. doi:10.1109/ACCESS.2021.3096189

[47] Caroline Kim, John Y. Kim, and Dr. Guofang Wei. 2008. Invertibility Probability of Binary Matrices Team. In *Team.* https://api.semanticscholar.org/CorpusID:15037510

[48] Brent Lagesse, Gabriel Nguyen, Utsav Goswami, and Kevin Wu. 2021. 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*. IEEE, 730–735. doi:10.1109/PERCOMWORKSHOPS51409.2021.9431029

[49] Duc-Phong Le, Sze Ling Yeo, and Khoongming Khoo. 2021. Algebraic Differential Fault Analysis on SIMON block cipher. *IACR Cryptol. ePrint Arch.* (2021), 436. https://eprint.iacr.org/2021/436

[50] Baiyu Li and Daniele Micciancio. 2021. On the Security of Homomorphic Encryption on Approximate Numbers. In *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12696)*, Anne Canteaut and François-Xavier Standaert (Eds.). Springer, 648–677. doi:10.1007/978-3-030-77870-5_23

[51] Fukang Liu, Santanu Sarkar, Willi Meier, and Takanori Isobe. 2021. Algebraic Attacks on Rasta and Dasta Using Low-Degree Equations. In *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 13090)*, Mehdi Tibouchi and Huaxiong Wang (Eds.). Springer, 214–240. doi:10.1007/978-3-030-92062-3_8

[52] Patrick Longa and Michael Naehrig. 2016. Speeding up the Number Theoretic Transform for Faster Ideal Lattice-Based Cryptography. In *Cryptology and Network Security*, Sara Foresti and Giuseppe Persiano (Eds.). Springer International Publishing, Cham, 124–139.

[53] Pei Luo, Cheng Li, and Yunsi Fei. 2016. Concurrent Error Detection for Reliable SHA-3 Design. In *Proceedings of the 26th edition on Great Lakes Symposium on VLSI, GLVLSI 2016, Boston, MA, USA, May 18-20, 2016*, Ayse K. Coskun, Martin Margala, Laleh Behjat, and Jie Han (Eds.). ACM, 39–44. doi:10.1145/2902961.2902985

[54] Paolo Maistri and Jiayun Po. 2022. A Low-Cost Methodology for EM Fault Emulation on FPGA. In *2022 Design, Automation & Test in Europe Conference & Exhibition, DATE 2022, Antwerp, Belgium, March 14-23, 2022*, Cristiana Bolchini, Ingrid Verbauwhede, and Elena-Ioana Vatajelu (Eds.). IEEE, 1185–1188. doi:10.23919/DATE54114.2022.9774507

[55] Pierrick Méaux, Claude Carlet, Anthony Journault, and François-Xavier Standaert. 2019. Improved Filter Permutators for Efficient FHE: Better Instances and Implementations. In *Progress in Cryptology - INDOCRYPT 2019 - 20th International Conference on Cryptology in India, Hyderabad, India, December 15-18, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11898)*, Feng Hao, Sushmita Ruj, and Sourav Sen Gupta (Eds.). Springer, 68–91. doi:10.1007/978-3-030-35423-7_4

[56] Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. 2016. Towards Stream Ciphers for Efficient FHE with Low-Noise Ciphertexts. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 9665)*, Marc Fischlin and Jean-Sébastien Coron (Eds.). Springer, 311–343. doi:10.1007/978-3-662-49890-3_13

[57] Pierrick Méaux and Dibyendu Roy. 2024. Theoretical differential fault attacks on FLIP and FiLIP. *Cryptogr. Commun.* 16, 4 (2024), 721–744. doi:10.1007/S12095-024-00698-Y

[58] Alexandre Menu, Jean-Max Dutertre, Olivier Potin, Jean-Baptiste Rigaud, and Jean-Luc Danger. 2020. Experimental Analysis of the Electromagnetic Instruction Skip Fault Model. In *15th Design & Technology of Integrated Systems in Nanoscale Era, DTIS 2020, Marrakech, Morocco, April 1-3, 2020*. IEEE, 1–7. doi:10.1109/DTIS48698.2020.9081261

[59] Ahmet Can Mert, Aikata, Sunmin Kwon, Youngsam Shin, Donghoon Yoo, Yongwoo Lee, and Sujoy Sinha Roy. 2023. Medha: Microcoded Hardware Accelerator for computing on Encrypted Data. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2023, 1 (2023), 463–500. doi:10.46586/tches.v2023.i1.463-500

[60] Debdeep Mukhopadhyay. 2009. An Improved Fault Based Attack of the Advanced Encryption Standard. In *Progress in Cryptology - AFRICACRYPT 2009, Second International Conference on Cryptology in Africa, Gammarth, Tunisia, June 21-25, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5580)*, Bart Preneel (Ed.). Springer, 421–434. doi:10.1007/978-3-642-02384-2_26

[61] Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. 2011. Can homomorphic encryption be practical?. In *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011*, Christian Cachin and Thomas Ristenpart (Eds.). ACM, 113–124. https://dl.acm.org/citation.cfm?id=2046682

[62] Chao Niu, Benqiang Wei, Zhicong Huang, Zhaomin Yang, Cheng Hong, Meiqin Wang, and Tao Wei. 2025. SoK: FHE-Friendly Symmetric Ciphers and Transciphering. IACR Trans. Cryptogr. Hardw. Embed. Syst. https://eprint.iacr.org/2025/669

[63] G. Bertoni, J. Daemen, M. Peeters and G. Van Assche. 2011. The Keccak reference. Round 3 submission to NIST SHA-3. http://keccak.noekeon.org/Keccak-reference-3.0.pdf

[64] R. Radheshwar, Meenakshi Kansal, Pierrick Méaux, and Dibyendu Roy. 2023. Differential Fault Attack on Rasta and FiLIP_{DSM}. *IEEE Trans. Computers* 72, 8 (2023), 2418–2425. doi:10.1109/TC.2023.3244629

[65] Dibyendu Roy, Bhagwan N. Bathe, and Subhamoy Maitra. 2021. Differential Fault Attack on Kreyvium & FLIP. *IEEE Trans. Computers* 70, 12 (2021), 2161–2167. doi:10.1109/TC.2020.3038236

[66] Mark Rudelson. 2006. Norm of the inverse of a random matrix. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*. IEEE Computer Society, 487–496. doi:10.1109/FOCS.2006.52

[67] Dhiman Saha, Sukhendu Kuila, and Dipanwita Roy Chowdhury. 2014. EscApe: Diagonal Fault Analysis of APE. In *Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India, New Delhi, India, December 14-17, 2014, Proceedings (Lecture Notes in Computer Science, Vol. 8885)*, Willi Meier and Debdeep Mukhopadhyay (Eds.). Springer, 197–216. doi:10.1007/978-3-319-13039-2_12

[68] Dhiman Saha, Debdeep Mukhopadhyay, and Dipanwita Roy Chowdhury. 2009. A Diagonal Fault Attack on the Advanced Encryption Standard. *IACR Cryptol. ePrint Arch.* (2009), 581. http://eprint.iacr.org/2009/581

[69] Xhani Marvin Saß, Richard Mitev, and Ahmad-Reza Sadeghi. 2023. Oops..! I Glitched It Again! How to Multi-Glitch the Glitching-Protections on ARM TrustZone-M. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 6239–6256. https://www.usenix.org/conference/usenixsecurity23/presentation/sass

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

[71] Florian Tramèr, Dan Boneh, and Kenny Paterson. 2020. Remote Side-Channel Attacks on Anonymous Transactions. In *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, Srdjan Capkun and Franziska Roesner (Eds.). USENIX Association, 2739–2756. https://www.usenix.org/conference/usenixsecurity20/presentation/tramer

[72] Michael Tunstall, Debdeep Mukhopadhyay, and Subidh Ali. 2011. Differential Fault Analysis of the Advanced Encryption Standard Using a Single Fault. In *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication - 5th IFIP WG 11.2 International Workshop, WISTP 2011, Heraklion, Crete, Greece, June 1-3, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6633)*, Claudio A. Ardagna and Jianying Zhou (Eds.). Springer, 224–233. doi:10.1007/978-3-642-21040-2_15

[73] Yingchen Wang, Riccardo Paccagnella, Elizabeth Tang He, Hovav Shacham, Christopher W. Fletcher, and David Kohlbrenner. 2022. Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86. In *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, Kevin R. B. Butler and Kurt Thomas (Eds.). USENIX Association, 679–697. https://www.usenix.org/conference/usenixsecurity22/presentation/wang-yingchen

Appendix

A Authenticated Transciphering (AT)

In the homomorphic setting, AT [1, 7] 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_{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 4.

First, AT was analyzed by the authors in [7]. They utilized Grain128-AEAD [44], and this was converted to a TFHE [20] ciphertext on the server side. This is followed by a recent work [1], where the authors report benchmarks for authenticated transciphering using standard symmetric encryption schemes such as AES-GCM and ASCON for CKKS [19] 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.

B The IND-CPA^{D} Security Overview

The IND-CPAD security game is shown in Fig.10.

Definition (IND-CPA^{D} Security Game). Let $\prod$ = (KeyGen, Enc, Dec, Eval) be an FHE scheme with plaintext space $\mathcal{M}$ and ciphertext space C. For an experiment parameterized by a bit $b \in \{0, 1\}$ and

Table 4: A simplified algorithmic description of the Authenticated Transciphering protocol [1, 7].

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

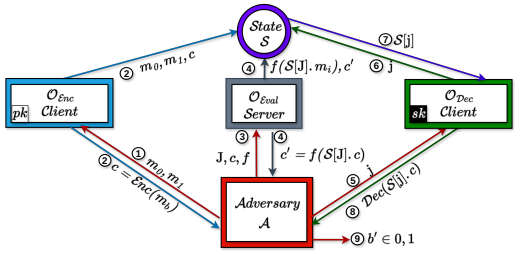

Figure 10: The IND-CPA^{D} threat model for FHE **[50].**

an adversary $\mathcal{A}$ that is given access to the following oracles O_{Enc} (Encryption Oracle), O_{Eval} (Evaluation Oracle), and O_{Dec} (Decryption Oracle) sharing a common state $S \in (\mathcal{M} \times \mathcal{M} \times C^*)$ consisting of message-message-ciphertext triplet sequence:

- Enc: $\mathcal{A}$ submits a pair of plaintext messages (m_0, m_1), to O_{Enc} which randomly chooses a secret bit b (0 or 1), and computes $c_{\text{FHE}} \leftarrow \text{Enc}_{\text{pk}}(m_b)$ 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 $f : \mathcal{M}^k \rightarrow \mathcal{M}$ and a sequence of indices $J = (j_1, \cdots, j_k) \in \{1, \cdots, |S|\}^k$ to O_{Eval} which computes the ciphertext $c_{\text{FHE}} \leftarrow \text{Eval}_{\text{pk}}(f, S[j_1] \cdot c_{\text{FHE}}, \cdots, S[j_k] \cdot c_{\text{FHE}})$, 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 O_{Dec} who checks whether $S[j] \cdot m_0 = S[j] \cdot m_1$, and, if so, returns $\text{Dec}_{\text{sk}}(S[j] \cdot c_{\text{FHE}})$, otherwise $\perp$ to the adversary.
- Guessing Stage: After multiple such queries, $\mathcal{A}$ tries to guess b. If correct, $\mathcal{A}$ wins; otherwise, she loses.

It should be noted –

Table 5: 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 $\{sK, pK\}$ $K_{FHE} = \mathsf{FHE}_{Enc}(K)_{pK}$ Send pK and K_{FHE}	$\rightarrow$	$\rightarrow$ Save pK and K_{FHE}
Encryption using SE		
$c_{i_{SE}} = m_i + \mathsf{E}_\pi(K, n_i)$ Send $c_{i_{SE}}$, n_i, and f	$\rightarrow$	$\rightarrow$ $c_{i_{SE}}$, n_i, and f
Homomorphic SE *Decryption Circuit Evaluation (*HSD*)*		
		$c_{i_{FHE}} = c_{i_{SE}} - \mathsf{FHE}_{E_\pi}(n_i, K_{FHE})$
Function Evaluation		
		Compute $d_{i_{FHE}} = f(c_{i_{FHE}})$
$d_{i_{FHE}}$	$\leftarrow$	$\leftarrow$ Send $d_{i_{FHE}}$
Result Decryption		
$f(m_i) = \mathsf{FHE}_{Dec}(d_{i_{FHE}})_{sK}$		

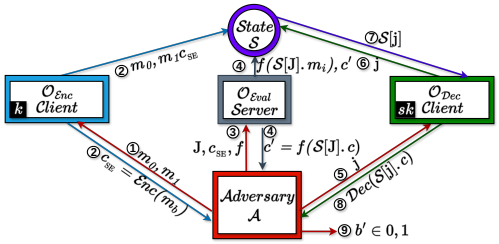

Figure 11: The IND-CPA^{D} threat model [50] for HHE.

- 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.

B.1 IND-CPA^{D} Security for HHE.

In the context of HHE, O_{Enc} performs symmetric encryption, and 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^{D} security game, as shown in Fig. 11.

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

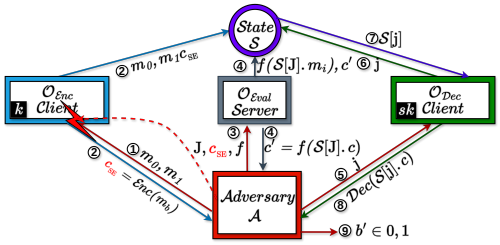

Figure 12: The adaptation of IND-CPA^{D} threat model [50] for HHE to the SASTA-DFA model, which extends it by incorporating fault injection capability.

B.2 IND-CPA^{D} Security for HHE in the presence of a Fault.

While prior works [16, 17, 50] could attack FHE schemes under the IND-CPA^{D} security notion, these attacks can be mitigated using noise flooding or rounding [50]. Thus, in the setting where the IND-CPA^{D} attacks are mitigated (or for the FHE schemes [20] unexploitable under this model), we explore if we can mount a 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 B.1) are as follows (Fig. 12).

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

Note that due to the use of nonce, every $\mathsf{E}_\pi(K, n_i)$ is distinct, and the adversary does not know the current value of $\mathsf{E}_\pi(K, n_i)$. 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 $\mathsf{E}_\pi(K, n_i)$ 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 during the non-Key-dependent phase (e.g., the XOF function targeted in this work).

B.3 IND-CPA^{D} Security for AT in the presence of a Fault.

We formalize the security of the AT variant under faulty executions by an IND-CPA-style distinguishing game that explicitly accounts for the additional values exposed by the AT verification flow. Concretely, in AT, the client decrypts the server-computed authentication tag to compare it with the locally computed tag; thus, under

faults, it is meaningful to model an adversary that obtains both tags from a single execution. We adapt the HHE model to AT as follows:

- Enc: $\mathcal{A}$ submits a pair of plaintext messages (m_0, m_1), associated data AD, and a nonce n_i, to O_{Enc} which randomly chooses a secret bit b (0 or 1), and computes Tag and Ciphertext c_{SE}. The adversary induces a fault during the computation of the first $\mathsf{E}_\pi(K, n_i)$, which is used only for tag computation, thus resulting in faulty tag $\mathtt{Tag}' \leftarrow ADE_i + \mathsf{E}'_\pi(K, n_i)$.
- Eval: $\mathcal{A}$ submits a function (or circuit) f to O_{Eval}. The oracle homomorphically evaluates f on the challenge ciphertext and returns the resulting ciphertext $c^f_{\text{FHE}} \leftarrow \mathsf{FHE.Eval}(pk, f, c_{\text{SE}})$ to the client, which decrypts to obtain $y \leftarrow \mathsf{FHE.Dec}(sk, c^f_{\text{FHE}})$. (Unlike HHE game, f can be any arbitrary function.)
- Tag: In addition, the server homomorphically computes the authentication tag for the ciphertext and returns $\mathtt{Tag}_{\text{FHE}} \leftarrow \mathsf{FHE.Eval}(pk, \text{TagAlg}, c_{\text{SE}}, AD)$, which the client decrypts to obtain the server-computed $\mathtt{Tag} \leftarrow \mathsf{FHE.Dec}(sk, \mathtt{Tag}_{\text{FHE}})$.
- Guess: $\mathcal{A}$ outputs a bit b'. The experiment outputs 1 if $b' = b$ and 0 otherwise.

This definition differs from the fault-free AT notion only in that the adversary is explicitly provided with both the client-computed tag- $\mathtt{Tag}'$ and the server-computed tag- $\mathtt{Tag}$, which are naturally available to the client during tag verification. Consequently, under faults, the adversary may exploit the correlation between these values even when f is arbitrary.

C Extending SASTA-DFA to AT

As described in Appendix A, AT is used to ensure data integrity in HHE. This is important to guarantee that the result obtained by the client is for the intended data and not faulty data. In [1, 7], 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 (E_π) of any new encryption since it is only used for tag generation and does not affect the subsequent ciphertext generation (as shown in Fig. 14). A stepwise description is provided in Table 6. 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.

Note that our attack does not alter or influence the associated data absorption and encryption part, shown as a grey box in Fig. 14. 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 6 (i)).

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

Table 6: Faulty computation flow during AT.

Client	Server
Initialization	
Encryption using SE	
$\acute{m}_i = m_i + \mathsf{E}_\pi(K, n_i)$	
$ADE_i = GCM_{partial}(\mathsf{E}_\pi, \acute{m}_i, K)$	
$\mathtt{Tag}'_i = \mathsf{E}'_\pi(K, n_i) \oplus ADE_i$ (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 $\mathtt{Tag}'_i == \mathsf{FHE}_{Dec}(\mathtt{Tag}_{i_{FHE}})_{sK}$: (ii) $f(m_i) = \mathsf{FHE}_{Dec}(d_{i_{FHE}})_{sK}$	

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

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

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 6 (ii)).

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

The differential thus obtained 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.

D 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. 13 (a). This is illustrated in the following example.

Consider a client-server interaction as shown in Fig. 13 (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 [64, 65] 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 [39]).

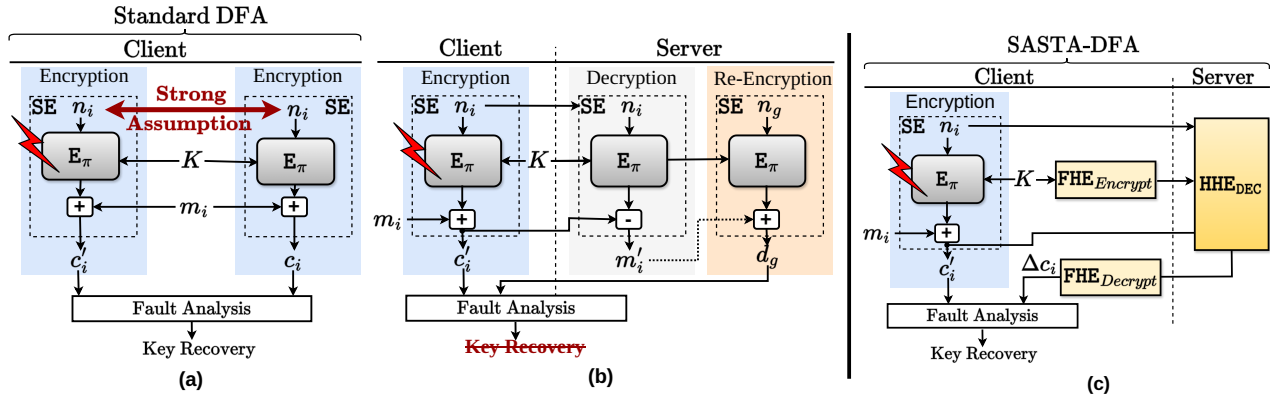

Figure 13: The diagram illustrates the difference between the standard DFA model (a) and the proposed SASTA-DFA model (c), showcasing how the classical setting relies on nonce-reuse, unlike SASTA-DFA. It also presents a case for standard DFA with server interaction (b), indicating how key recovery is infeasible as per specification, $n_i \neq n_g$.

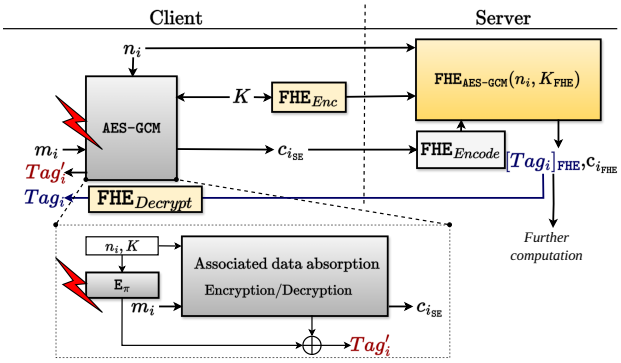

Figure 14: Exploiting SASTA-DFA for Authenticated Transciphering using AES-GCM.

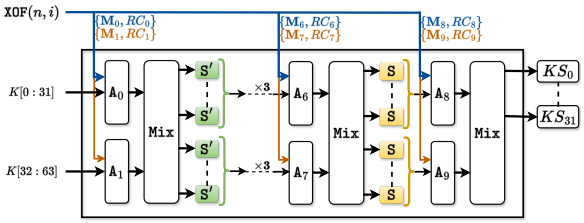

Figure 15: The PASTA-4 encryption takes as input the key (K), nonce (n), and counter (i), and ultimately generates the truncated result- key stream (KS).

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. 13 (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 E_π), 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 E_π. Thus, SASTA-DFA enables an attacker to perform key recovery without replaying plaintext or any other public parameter, including the nonce.

E PASTA scheme design overview

In this section, we will give a brief idea of new SE schemes design. For this, we choose PASTA [27] due to its comprehensive support for various operations. Other designs can be viewed as adaptations or variations of PASTA and are detailed in Appendices F, G, and H. In the context of HHE, variables can be categorized into two types: public and private. As illustrated in Fig. 2 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.

- A_i (Affine Layer): For this layer, an *invertible* matrix $\mathbf{M}_i$ and a round constant vector RC_i are generated using the SHAKE128 XOF output. Then, the layer performs $\mathbf{M}_i \cdot X_i + RC_i$

operation, where X_i represents the input state comprising t coefficients.

- `Mix` (Mixing Layer): Following $\mathtt{A}_i$, the two halves of the state are mixed using the `Mix` operation. This operation transforms the state into $(2 \cdot X_L + X_R, 2 \cdot X_R + X_L)$. This step is crucial for spreading values evenly across the two-state halves.
- $\mathtt{S}'/\mathtt{S}$ (S-Box Layer): The next layer involves the S-Box operation. For the final round, the cube S-Box ($\mathtt{S}$) is applied, while for all previous rounds, a Feistel S-Box ($\mathtt{S}'$) is utilized. Both these s-boxes are *invertible*.
- 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.

F Masta and Rasta design overview

Masta [42] 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. 16) 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 [24] is similar to Masta with the only difference being operation over $\mathbb{Z}_2$ and the choice of S-box.

$$\mathtt{S}_\chi(X^j) = X^j + X^{j+2} + X^{j+1} \cdot X^{j+2} \pmod{p} \tag{16}$$

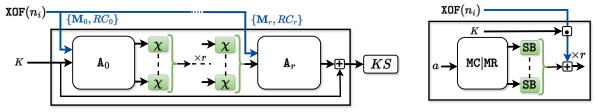

Figure 16: The r-round Masta and a general idea of Hera and Rubato round functions.

G Hera design overview

Hera [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).

H Rubato design overview

Rubato [43] 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.

I Keccak and Number Theoretic Transform

The Keccak [63] 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) [52, 70] facilitates the conversion of polynomials from coefficient representation to slot representation. With this transformation, polynomial multiplication has a cheap $O(n \log n)$ complexity, in contrast to the expensive $O(n^2)$ 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.

J 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. 17 [40, 41]. This ensures that the matrix is invertible

$$\mathbf{M}_i^{j+1} = \mathbf{M}_i^j \cdot \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ . & . & . & \cdots & . \\ . & . & . & \cdots & . \\ 0 & 0 & 0 & \cdots & 1 \\ \alpha_0 & \alpha_1 & \alpha_2 & \cdots & \alpha_{t-1} \end{bmatrix} \forall\ 1 \le j < t \tag{17}$$

K 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

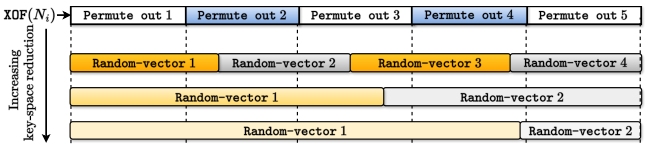

Figure 17: This figure shows that more KECCAK permutations for random vector generation of HERA make it easier to target a specific vector without affecting others, leading to a higher reduction in the keyspace.

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.

L 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 L.1 (**THE MIRROR EFFECT**). *The* SASTA-DFA *fault model can be used on the server side during* HSD *to obtain the differential:*

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

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 J. 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 **M**, or the vector in NTT form (discussed in Appendix I). 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 [11, 14, 28, 58]. However, attacking the server-side computation can be challenging due to limited opportunities and constraints.

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 and F for faults. in [5].

Target Scheme	Sec. bits	Field	Nonce Resp.	#F	Time	HHE	Ref.
Flip_{530}	80	$\mathbb{Z}_2$	No	1	39hr	✗	[65]
Kreyvium	128	$\mathbb{Z}_2$		3	5min	✗	[65]
Filip_{430}	80	$\mathbb{Z}_2$		1	751hr	✗	[64]
RASTA-6	80	$\mathbb{Z}_2$		1	96hr	✗	[64]
RASTA-5	128	$\mathbb{Z}_2$	**No**	1	0.11s	✗	Sec 4.5
RASTA-6	128	$\mathbb{Z}_2$		1	0.12s	✗	Sec 4.5
PASTA-3	128	$\mathbb{F}_p$		1	0.50s	✗	Sec 4.1
PASTA-4	128	$\mathbb{F}_p$		1	0.21s	✗	Sec 4.1
MASTA-4	128	$\mathbb{F}_p$		1	0.09s	✗	Sec 4.2
MASTA-5	128	$\mathbb{F}_p$		1	0.07s	✗	Sec 4.2
HERA-5	128	$\mathbb{F}_p$		1	0.005s	✗	Sec 4.3
RUBATO	128	$\mathbb{F}_p$		2	0.15s	✗	Sec 4.4
AES-GCM	128	$\mathbb{Z}_2$	**Yes**	1	5s	✓	Sec 4.6
AES-GCM	256	$\mathbb{Z}_2$		1	45min†	✓	Sec 4.6
RASTA-5	128	$\mathbb{Z}_2$		1	0.11s	✓	Sec 4.5
RASTA-6	128	$\mathbb{Z}_2$		1	0.12s	✓	Sec 4.5
PASTA-3	128	$\mathbb{F}_p$		1	0.50s	✓	Sec 4.1
PASTA-4	128	$\mathbb{F}_p$		1	0.21s	✓	Sec 4.1
MASTA-4	128	$\mathbb{F}_p$		1	0.09s	✓	Sec 4.2
MASTA-5	128	$\mathbb{F}_p$		1	0.07s	✓	Sec 4.2
HERA-5	128	$\mathbb{F}_p$		1	0.05s	✓	Sec 4.3
RUBATO	128	$\mathbb{F}_p$		2	0.15s	✓	Sec 4.4

Although the fault can be introduced on both the client and server side (Corollary L.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.

M 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.