# Single Trace Side-Channel Attack on the MPC-in-the-Head Framework

Julie Godard<sup>1,2</sup>, Nicolas Aragon<sup>1</sup>, Philippe Gaborit<sup>1</sup>, Antoine Loiseau<sup>2</sup> and Julien Maillard<sup>1,2</sup>

XLIM, University of Limoges, Limoges
 <sup>2</sup> Univ. Grenoble Alpes, CEA, Leti, F-38000 Grenoble, France

Abstract. In this paper, we present the first single trace side-channel attack that targets the MPC-in-the-Head (MPCitH) framework based on threshold secret sharing, also known as Threshold Computation in the Head (TCitH) in its original version. This MPCitH framework can be found in 5 of the 14 digital signatures schemes in the recent second round of the National Institute of Standards and Technology (NIST) call for digital signatures. In this work, we start by highlighting a side-channel vulnerability of the TCitH framework and show an exploitation of it on the SDitH algorithm, which is part of this NIST call. Specifically, we exploit the leakage of a multiplication function in the Galois field to make predictions about intermediate values, and we use the structure of the algorithm to combine information efficiently. This allows us to build an attack that is both the first Soft Analytical Side-Channel Attack (SASCA) targeting the MPCitH framework, as well as the first attack on SDitH. More specifically, we build a SASCA based on Belief Propagation (BP) on the evaluation of polynomials in the signature using the threshold variant structure to reconstruct the secret key. We perform simulated attacks under the Hamming Weight (HW) leakage model, enabling us to evaluate the resistance of the scheme against SASCA. We then perform our attacks in a real case scenario, more specifically on the STM32F407, and recover the secret key for all the security levels. We end this paper by discussing the various shuffling countermeasures we could use to mitigate our attacks.

**Keywords:** Soft Analytical Side-Channel Attack (SASCA)  $\cdot$  Belief Propagation (BP)  $\cdot$  Syndrome Decoding in the Head (SDitH)  $\cdot$  Post-Quantum Cryptography (PQC)  $\cdot$  Key recovery  $\cdot$  Multi-Party Computation-in-the-Head (MPCitH)

# Introduction

The MPC-in-the-Head framework, introduced in 2007 by [IKOS07], is a technique for constructing zero-knowledge proofs. This framework provides a generic way of constructing zero-knowledge proofs using secure multiparty computation techniques. For nearly a decade, this result was considered theoretical. Then, in 2016, Giacomelli et al. presented the first practical proof system based on this framework in [GMO16]. Since then this framework has been more widely studied and many algorithms based on this approach have been proposed, like MQOM [BFR23] or SDitH [AM<sup>+</sup>23].

Over the last few years, researchers have made new improvements to the MPC-in-the-Head (MPCitH) framework, leading to renewed interest in it. This interest is illustrated in particular by the new call from the National Institute of Standards and Technology (NIST) for digital signature schemes, in which 7 of the 40 candidates are based on this framework. More recently, NIST announced the second round, in which only 14 of candidates were retained, including 5 candidates based on the MPCitH framework, and in particular, the SDitH algorithm  $[AM^+23]$ . It is important to note that in the following, the attack we are going to describe is based on the implementation provided for the first round submission, as the revision for the second round has not been released yet.

The attacks we are proposing in this paper are Soft Analytical Side-Channel Attacks (SASCA), first introduced in [VGS14], and are effective methods to combine outputs of supervised attacks [CRR02] on mathematically linked intermediate variables. In [VGS14], the authors present the case of a practical attack on the AES Furious implementation, using the Belief Propagation (BP) algorithm [Pea82], and demonstrated the efficiency of SASCA compared to the best state-of-art attacks. As redundancy is a key components of the MPCitH protocol, and particularly in SDitH, SASCA is a relevant approach.

SASCA has also been applied to post-quantum algorithms. For example, the standardised Module-Lattice KEM (ML-KEM) algorithm [RL23], also known as Kyber, has been the target of several attacks of this type [PPM17], [PP19], [HHP<sup>+</sup>21] targeting the Number Theoretic Transform (NTT). We can also take the example of the HQC algorithm with [GMGL24] and [BMG<sup>+</sup>24]. More precisely, in [GMGL24], practical attacks against the Reed-Solomon (RS) decoder are presented. In these attacks, it is more specifically multiplication in the Galois field that the authors target in order to recover the shared key. Among others, in this paper, Goy et al. simulate an attack on the Reed-Solomon (RS) decoder using a Hamming Weight (HW) leakage model. This simulation gives a success rate of more than 0.9 up to a noise parameter  $\sigma = 2$ , and even up to  $\sigma = 3$  for the highest HQC security level. In practice, this attack has a 100% success rate. Their work also presents an analysis of several countermeasures, including a full shuffling strategy that adds satisfactory combinatorial complexity to the proposed attacks.

**Our contributions** In this paper, we present the first side-channel attack against the TCitH framework, in its 2022 version [FR22], which is the one that was current at the time of the first round of the second NIST call. More specifically, we apply our attack to the SDitH algorithm and target its secret key. Our contributions are the following:

- We start by highlighting a vulnerability in the TCitH framework. Firstly, we target a function executed by all the parties and that manipulates a part of the secret key. This part is therefore dependent on the implementation. Secondly, we look at how the inputs of the parties (i.e., the shares of the secret key), involved in the MPC protocol, are constructed in order to find a link between these inputs and the secret key. This part is independent of the implementation.

- We analyze the vulnerability from a side-channel perspective. To do so, we start by applying it to the SDitH algorithm. In this algorithm, the function on which we focus is the polynomial evaluation using the gf\_mul function. We begin our analysis by performing leakage assessment on the gf\_mul operands and output on a STM32F407 platform. Then, with evidence of leakage, we build a template attack on these values and discuss the obtained accuracies.
- To make full use of the redundancy employed in SDitH, we conduct a SASCA. We use simulations to evaluate the resistance of the different security levels of SDitH against SASCA. Then we perform a practical attack that leads to a successful secret key recovery. In practice, this attack led to a perfect success rate for the three security levels.

**Outline** Section 1 recalls how the MPC-in-the-Head and the TC-in-the-Head frameworks work and describes the construction of the SDitH algorithm. Section 2 introduces the point of vulnerability of the TCitH framework, how we applied it to SDitH and introduce how SASCA works. Section 3 presents the attack on SDitH in theory with the template on the Galois field multiplication and the construction of the graph for the SASCA with some simulations. Section 4 shows how to perform our attack in practice and the different results that we obtain. Section 5 discusses shuffling countermeasures. Finally, Section 6 contains some conclusions about our attack, as well as some ideas for future work.

# **1** Preliminaries

Signature schemes can be built from an identification scheme, which corresponds to a zero-knowledge proof of knowledge of a secret key. The MPCitH paradigm itself creates a link between the notions of multiparty computation (MPC) and zero-knowledge protocol.

In this section, we introduce the notions of MPC protocols and the MPCitH paradigm. We then focus on the SDitH algorithm and in particular its threshold variant.

### 1.1 MPC protocols and the MPC-in-the-Head paradigm

A Multi-Party Computation (MPC) protocol is an interactive protocol allowing two or more parties to jointly compute a function on their private data without revealing this data to the other parties. Before 2022, when implementing MPC-based algorithms, it was necessary to emulate all the parties, say N, of the underlying MPC protocol. But, due to the renewed of interest in this framework, several optimisations have emerged. Starting in 2022, two optimisations were released in parallel, with the aim of reducing the number of MPC protocol emulations by modifying the generation of inputs (known as "shares") of the parties for the MPC protocol.

**Hypercube variant.** Let's start by looking at the first variant called the hypercube and proposed by [MGH<sup>+</sup>22]. This approach generates party shares using the hypercube structure, allowing the emulation of  $1 + \log_2(N)$  parties instead of N, this without any additional communication costs.

**Threshold variant.** There is also the Threshold Computation in the Head (TCitH) framework proposed in [FR22]. In this variant, shares are generated using a linear secret sharing scheme (LSSS). This comes at the cost of more communications due to the use of Merkle tree (due to the larger size of the hash digests in the authentication path). In addition, the number N of parties is limited by the size of the field in which the secret lies.

**Other variants.** One year later, in [FR23], the TCitH framework is improved in several ways. It includes, among other things, a solution for using GGM trees that removes the additional cost in terms of communication due to the use of the Merkle tree. At the same time as the work in [FR23] began, the VOLEin-the-Head framework [BBdSG<sup>+</sup>23] was published. According to [FR23] this framework can be seen as a sub-case of the TCitH framework for l = 1 and with the use of GGM trees except for a few exceptions (which we will not detail in this paper).

#### 1.1.1 Definitions

Before taking a closer look to SDitH, we provide some necessary vocabulary and definitions.

**Zero-knowledge proofs.** In a zero-knowledge proof, one party (the prover) wants to convince another party (the verifier) that he knows a statement, like the private key, without revealing anything about that statement.

The following definitions are based on the paper written by Feneuil and Rivain [FR22]:

**Threshold LSSS.** Let  $\mathbb{F}$  be a finite field and let  $\mathbb{V}_1$  and  $\mathbb{V}_2$  be two vector spaces over  $\mathbb{F}$ . Let t and N be integers such that  $1 < t \leq N$ . A (t, N)-threshold linear secret sharing scheme (LSSS) is a method to share a secret  $s \in \mathbb{V}_1$  into N shares  $[\![s]\!] := ([\![s]\!]_1, \ldots, [\![s]\!]_N) \in \mathbb{V}_2$  such that the secret can be reconstructed from any t shares while no information is revealed on the secret from the knowledge of t-1 shares.

Formally, a (t, N)-threshold LSSS consists of a pair of algorithms:

$$\begin{cases} Share : \mathbb{V}_1 \times R \mapsto \mathbb{V}_2^N \\ Reconstruct_J : \mathbb{V}_2^t \mapsto \mathbb{V}_1 \end{cases}$$

where  $R \subseteq \{0,1\}^*$  denotes some randomness space and where  $Reconstruct_J$  is indexed by a set (and defined for every)  $J \subset [N]$  such that |J| = t.

Shamir's Secret Sharing. The Shamir's Secret Sharing over  $\mathbb{F}$  is an (l+1, N)-threshold LSSS for which a sharing  $[\![s]\!]$  of  $s \in \mathbb{F}$  is constructed as follows:

- sample  $r_1, \ldots, r_l$  uniformly in  $\mathbb{F}$ ,
- build the polynomial P as  $P(X) := s + \sum_{i=1}^{l} r_i X^i$ ,
- build the shares  $[\![s]\!]_i$  as evaluations  $P(e_i)$  of P for each  $i \in \{1, \ldots, N\}$ , where  $e_1, \ldots, e_n$  are non-zero public distinct points of  $\mathbb{F}$ .

For any subset  $J \subseteq [N]$ , s.t. |J| = l + 1, the *Reconstruct*<sub>J</sub> algorithm interpolates the polynomial P from the l+1 evaluation points  $[\![s]\!]_J = (P(e_i))_{i \in J}$  and outputs the constant term s.

### 1.1.2 MPC Protocols

An MPC protocol is a protocol in which, given a statement x and a relation  $\mathcal{R}$ , N parties  $\mathcal{P}_i$  securely and correctly evaluate a function f on a secret sharing  $\llbracket w \rrbracket_i$  of the secret witness w for a statement x to check whether  $(x, w) \in \mathcal{R}$  or not. When  $(x, w) \in \mathcal{R}$ , the parties output ACCEPT, otherwise they output REJECT.

In this MPC model, parties are required to perform only linear operations on the shares. We can also note that this protocol is l-private, meaning that one can open l views (composed of its input share, its random tape and all its received messages) without getting any private information.

During the MPC protocol steps, parties can perform three types of action:

- **Receiving randomness:** the parties receive the same random value  $\epsilon$  from a randomness oracle  $\mathcal{O}_R$ .
- **Receiving hints:** the parties can receive a sharing of a hint  $\beta$  from a hint oracle  $\mathcal{O}_H$ . This hint may depend on the witness w and previously sent elements (such as random values sampled from  $\mathcal{O}_R$ ).
- Computing and broadcasting: Thanks to the linearity of sharing, the parties can perform linear transformations on their shares. That is, each party will be able to locally compute  $[\![\alpha]\!] := [\![\varphi(v)]\!]$  from a sharing  $[\![v]\!]$  where  $\varphi$  is an  $\mathbb{F}$ -linear function. Each of the parties  $\mathcal{P}_i$  will then be able to broadcast its share  $[\![\alpha]\!]_i$ , allowing public reconstruction  $\alpha := \varphi(v)$ .

We repeat these steps t times, and then we have that the publicly reconstructed values  $\alpha^1, \ldots, \alpha^t$  satisfy the relation  $g(\alpha^1, \ldots, \alpha^t) = 0$ , for a given function g, if and only if the parties output ACCEPT.

#### 1.1.3 MPC-in-the-Head Paradigm

Introduced by Ishai, Kushilevitz, Ostrovsky and Sahai in [IKOS07], the MPCin-the-Head paradigm makes it possible to construct a zero-knowledge proof of knowledge of a secret x using an MPC protocol. By "in the Head" we mean that the prover will emulate the MPC protocol locally. We can now explain how the zero-knowledge protocol works:

- 1. the prover generates the shares  $[x]_i$  of x and commits to them,
- 2. they receive a challenge from the verifier (the randomness  $\epsilon$ ),
- 3. thanks to this challenge, they simulate "in their head" all the parties  $\mathcal{P}_i$  of the MPC protocol,
- 4. they then send a commitment of the result of each party's MPC protocol (also called party's view),
- 5. they receive a second challenge to know which party's view they need to reveal,
- 6. they reveal the views,
- 7. and finally, the verifier can check the overall consistency of the MPC computation.

For a better understanding of this framework, we can also illustrate it as shown in **Figure 1**:

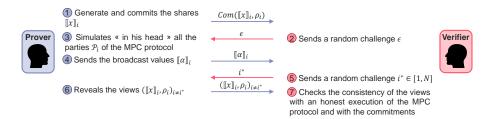


Fig. 1: Illustration of the MPC-in-the-Head framework.

### 1.1.4 MPC-in-the-Head with Threshold LSSS

In this section, we will look at the main changes that come from the application of a threshold LSSS (instead of a simple additive sharing system) to the MPCitH paradigm:

- the parties initially receive an (l+1, N)-threshold LSS of the witness w,
- when parties receive hints from the  $\mathcal{O}_H$  oracle, they take the form of an (l+1, N) threshold LSS sharing of  $\beta$ ,
- to reconstruct the value  $\alpha^{j}$  from their shares, they use the algorithm *Reconstruct*.

### 1.2 SD-in-the-Head

The SDitH signature scheme is based on the hardness of the syndrome decoding problem for random linear codes over a finite field. It uses an MPC protocol to check whether a given shared input corresponds to the solution of a syndrome decoding instance. By applying the MPC-in-the-Head paradigm to the protocol, it becomes a zero-knowledge proof of the knowledge of a low-weight vector x solution of an instance of the Syndrome Decoding (SD) problem. The Fiat-Shamir heuristic [FS86] can then be used to transform the interactive proof into a non-interactive proof and thus into a signature scheme. The SDitH protocol, originally proposed in [FJR22], has undergone improvements in subsequent works [MGH<sup>+</sup>22], [FR22] that gave rise to the hypercube and threshold variants used in SDitH.

In the following, we will refer to SDitH security levels I, III and V. NIST has defined these levels in [NIS16] as follows:

- Level I the algorithm is at least as hard to break as AES-128 using exhaustive key search.
- Level III the algorithm is at least as hard to break AES-192 using exhaustive key search.
- Level V the algorithm is at least as hard to break AES-256 using exhaustive key search.

The notations used in the rest of the paper are summarised in Table 1:

Table 1: Notations and parameters of the SDitH scheme from [AM<sup>+</sup>23].

Table 1. Retailed of the Spith Science from [Intr 20].						
Syndrome	Syndrome decoding parameters:					
m	Code length.					
k	Code dimension.					
w	Hamming weight bound.					
d	Parameter of the $d$ -splitting variant.					
Signature parameters:						
au	Number of repetitions.					
t	Number of random evaluation points.					
Field parameters:						
$\mathbb{F}_q$	Field with $q$ elements: base field of the SD instance					
$\mathbb{F}_{\text{points}}$	Extension field of $\mathbb{F}_q$ (base field of the MPC elements $\alpha, \beta, v, r, \epsilon$ )					
$\eta^{-}$	Field extension s.t. $\mathbb{F}_{\text{points}} = \mathbb{F}_{q^{\eta}}$ .					
MPC protocol parameters:						
l	Set of opened parties $(I \subseteq [1:N],  I  = l)$ .					
$r_{cols}$	Number of columns in the matrix composed of powers of points $r_k$ .					

### 1.2.1 Splitting Syndrome Decoding

Let us start by defining the problem on which the SDitH algorithm is based.

The syndrome decoding problem. Let  $\mathbb{F}$  a finite field, and m, k, w three positive integers such that  $m \geq k$  and  $m \geq w$ . An instance of the syndrome decoding problem with parameters  $(\mathbb{F}, m, k, w)$  consists of:

- a parity-check matrix in standard form  $H = (H'|I_{m-k}) \in \mathbb{F}^{(m-k) \times m}$ ,
- a vector  $y \in \mathbb{F}^{m-k}$ .

A solution to the problem is a vector  $x = (x_A | x_B) \in \mathbb{F}^m$  such that:

$$y := Hx = H'x_A + x_B$$
 and  $wt(x) \le w$ 

where we represent the Hamming weight. Using this representation for the matrix H and the vector x, we only need  $x_A$  to find the solution.

*d*-split syndrome decoding problem. This is a variant of the syndrome decoding problem where we split the solution x into d blocks  $x_1, \ldots, x_d \in \mathbb{F}_q^{m/d}$  such that:

$$x = (x_1|\ldots|x_d)$$
 s.t.  $\operatorname{wt}(x_j) = \frac{w}{d} \quad \forall j \in [1:d].$ 

This variant is used for security levels III and V.

# 1.2.2 The SDitH MPC Protocol

In the algorithm specification, in order to build the MPC protocol, it is necessary to use linear functions, and therefore to characterise the constraint  $wt(x) \leq w$  linearly. To do this, the polynomial representation of [AM<sup>+</sup>23] is used.

Let  $f_1, \ldots, f_q$  denote the elements of  $\mathbb{F}_q$ . We want to build three witnesspolynomials S, Q, and P, and one public polynomial F such that :

$$S \cdot Q = P \cdot F$$

These polynomials are defined as follows:

- The polynomial  $S \in \mathbb{F}_q[X]$  is obtained by Lagrange interpolation of the coordinates of x, such that  $S(f_i) = x_i$  for  $i \in [1 : m]$ . This polynomial is of degree  $\deg(S) \leq m 1$ .
- The polynomial  $Q \in \mathbb{F}_q[X]$  is defined as  $Q(X) = \prod_{i \in E} (X f_i)$ , where E is a subset of [1:m] of order |E| = w, such that the non-zero coordinates of x are contained in E. This polynomial is of degree  $\deg(Q) = w$ .
- The polynomial  $F \in \mathbb{F}_q[X]$  is the "vanishing polynomial" of the set  $f_1, \ldots, f_m$  which is defined as  $F(X) = \prod_{i \in [1:m]} (X f_i)$ . This polynomial is of degree  $\deg(F) = m$ .
- The polynomial  $P \in \mathbb{F}_q[X]$  is defined as  $P = S \cdot Q/F$ . This polynomial is of degree deg $(P) \leq w 1$ .

By constructing these polynomials, we have that  $S \cdot Q = P \cdot F = 0$  at all points  $f_i \in [1:m]$ . Then, at each point  $f_i$ , either  $S(f_i) = x_i = 0$ , or  $Q(f_i) = 0$ . But Q can be zero in at most w points because of its degree. And by construction of the set E, we have that S is non-zero in at most w points, meaning that  $wt(x) \leq w$  (more details can be found in [AM<sup>+</sup>23]).

When considering the *d*-split variant of the SD problem, the polynomials P and Q are replaced by the *d*-split vectors  $P = (P[1], \ldots, P[d])$  and  $Q = (Q[1], \ldots, Q[d])$  such that  $\deg(P[j]) \leq w/d$  and  $\deg(Q[j]) = w/d \quad \forall j \in [1:d]$ . We also consider the challenges  $r, \epsilon$ , and by construction the shares  $[\![\alpha]\!], [\![\beta]\!], [\![v]\!]$  in a *d*-vectorised form.

To check that the polynomial relation  $S \cdot Q = P \cdot F$  is true, we use the Schwartz-Zippel lemma, which states that the equality of polynomials is unlikely to be true at randomly chosen points if the equality is not true in general. And since the evaluation of a polynomial at a point is a linear operation, the parties will be able to locally compute the evaluations  $S(r_k)$ ,  $Q(r_k)$  and  $P - F(r_k)$  for each random point  $r_k$ . And to check the equality  $S(r_k) - Q(r_k) = P - F(r_k)$ , we use the protocol in [BN20].

SDitH's MPC protocol is therefore built as follows in [AM<sup>+</sup>23]:

- 1. Sample  $r, \epsilon \in \mathbb{F}_{points}^t$  uniformly at random.
- 2. Parties locally set  $\llbracket x_B \rrbracket = y H' \llbracket x_A \rrbracket$ .
- 3. Parties locally compute [S] via Lagrange interpolation of  $[x] = ([x_A] | [x_B])$ .
- 4. Parties locally evaluate  $[\![S(r_k)]\!]$ ,  $[\![Q(r_k)]\!]$  and  $[\![F \cdot P(r_k)]\!]$ .
- 5. For all  $j \in [t]$ , parties verify  $(\llbracket S(r_k) \rrbracket, \llbracket Q(r_k) \rrbracket, \llbracket F \cdot P(r_k) \rrbracket)$  by sacrificing the triple  $(\llbracket a_k \rrbracket, \llbracket b_k \rrbracket, \llbracket c_k \rrbracket)$ :
  - (a) Parties locally compute

$$[\![\alpha_k]\!] = \epsilon_k \cdot [\![Q(r_k)]\!] + [\![a_k]\!], \text{ and set } [\![\beta_k]\!] = [\![S(r_k)]\!] + [\![b_k]\!].$$

- (b) Parties broadcast  $\llbracket \alpha_k \rrbracket$  and  $\llbracket \beta_k \rrbracket$  to publicly recompute  $\alpha_k$  and  $\beta_k$
- (c) Parties locally compute

$$\llbracket v_k \rrbracket = \epsilon_k \cdot \llbracket F \cdot P(r_k) \rrbracket - \llbracket c_k \rrbracket + \alpha_k \cdot \llbracket b_k \rrbracket + \beta_k \cdot \llbracket a_k \rrbracket - \alpha_k \cdot \beta_k.$$

- (d) Parties broadcast  $\llbracket v_k \rrbracket$  to publicly recompute  $v_k$ .
- (e) Parties output Accept if  $v_k = 0$  and Reject otherwise.

When parties compute  $[S(r_k)]$  locally, this corresponds to a vector/matrix multiplication in the code. This multiplication is composed of the Galois field multiplication, hereafter referred to as gf\_mul.

### 1.2.3 The SDitH Signature and Verification Algorithms

Thanks to the MPCitH paradigm, we can transform the SDitH MPC protocol into a zero-knowledge proof, and then use the Fiat-Shamir heuristic to get the SDitH signature scheme. We start by giving the specifications of the signature scheme from  $[AM^+23]$ :

### Signature:

- 1. Generate random sharing  $[\![x_A]\!], [\![P]\!], [\![Q]\!], [\![a]\!], [\![b]\!], [\![c]\!]$
- 2. Commit the partie's shares:

$$\llbracket x_A \rrbracket_i, \llbracket P \rrbracket_i, \llbracket Q \rrbracket_i, \llbracket a \rrbracket_i, \llbracket b \rrbracket_i, \llbracket c \rrbracket_i \xrightarrow{\text{Commit}} \text{com}_i$$

3. Derive the first challenge (randomness of MPC protocol):

$$\operatorname{com}_1, \ldots, \operatorname{com}_N \xrightarrow{\operatorname{Hash}} h_1 \to r, \epsilon$$

4. Simulate the MPC protocol:

$$\llbracket x_A \rrbracket, \llbracket P \rrbracket, \llbracket Q \rrbracket, \llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket, r, \epsilon \xrightarrow{\text{MPC}} \llbracket \alpha \rrbracket, \llbracket \beta \rrbracket, \llbracket v \rrbracket$$

5. Derive the second challenge (index of the non-opened party):

$$h_1, \llbracket \alpha \rrbracket, \llbracket \beta \rrbracket, \llbracket v \rrbracket \xrightarrow{\text{Hash}} h_2 \to I$$

6. Build the signature from

$$h_1, h_2, \{ [\![x_A]\!]_i, [\![P]\!]_i, [\![Q]\!]_i, [\![a]\!]_i, [\![b]\!]_i, [\![c]\!]_i \}_{i \in I}, \{ \operatorname{com}_i, [\![\alpha]\!]_i, [\![\beta]\!]_i, [\![v]\!]_i \}_{i \notin I} \}_{i \notin I} \}_{i \in I} \}$$

We now do the same for the verification scheme from  $[AM^+23]$ :

# Verification:

1. Recompute the commitments, for parties  $i \in I$  (with I obtained from  $h_2$ ):

$$\llbracket x_A \rrbracket_i, \llbracket P \rrbracket_i, \llbracket Q \rrbracket_i, \llbracket a \rrbracket_i, \llbracket b \rrbracket_i, \llbracket c \rrbracket_i \xrightarrow{\text{Commit}} \operatorname{com}_i$$

2. Recompute the first challenge (randomness of MPC protocol):

$$\operatorname{com}_1, \ldots, \operatorname{com}_N \xrightarrow{\operatorname{Hash}} h_1 \to r, \epsilon$$

3. Simulate the MPC protocol, for parties  $i \in I$ :

$$\llbracket x_A \rrbracket_i, \llbracket P \rrbracket_i, \llbracket Q \rrbracket_i, \llbracket a \rrbracket_i, \llbracket b \rrbracket_i, \llbracket c \rrbracket_i, r, \epsilon \xrightarrow{\text{MPC}} \llbracket \alpha \rrbracket_i, \llbracket \beta \rrbracket_i, \llbracket v \rrbracket_i$$

4. Recompute the second challenge (index of non-opened party):

$$h_1, \llbracket \alpha \rrbracket, \llbracket \beta \rrbracket, \llbracket v \rrbracket \xrightarrow{\text{Hash}} h_2$$

5. Check that recomputed  $h_1, h_2$  match the signature.

### 1.3 The Threshold Variant

The SDitH algorithm uses the hypercube and threshold variants (the 2022 version). In light of the recent improvements, we have decided to focus our work on the threshold variant, particularly on this variant over the field  $\mathbb{F}_{256}$ .

In this variant, the Shamir's secret sharing is used as LSSS and the MPC protocol calculation is run on the l + 1 coefficients of the polynomial involved in sharing Shamir's secret. Noting:

input\_plain := 
$$(x_A, P, Q, a, b, c)$$
,

and sampling the l vectors  $\mathsf{input\_coef}_l, ..., \mathsf{input\_coef}_l$  uniformly at random in  $F_q^{|\mathsf{input\_plain}|}$ , we can define the  $i^{th}$  share of  $\mathsf{input\_plain}$  as follows:

$$\llbracket \mathsf{input\_plain} \rrbracket_i := \mathsf{input\_plain} + \sum_{j=1}^l f_i^j \cdot \mathsf{input\_coef}_j. \tag{1}$$

The MPC computation is then run on input\_plain, input\_coef<sub>1</sub>, ..., input\_coef<sub>l</sub>. Furthermore, in  $[AM^+23]$  it has been chosen to avoid calculating the Lagrange interpolation of the polynomial S, with the result that  $[s_A]$  can be given as input to the MPC protocol for the parties instead of  $[x_A]$ . The private variable of the private key is the following:

wit\_plain := 
$$(s_A, Q', P);$$

with Q' the truncated version of Q.

We end this subsection with the **Table 2** which summarises some parameters of the SDitH algorithm for the threshold variant:

Table 2. The fift offit parameters for the threshold variant from [fifth 20].									
	l	au	k	m	w	d	$\eta$	t	$r_{cols}$
SDitH-L1	3	6	126	242	87	1	4	7	32
SDitH-L3	3	9	220	376	114	2	4	10	48
SDitH-L5	3	12	282	494	156	2	4	13	64

Table 2: The MPCitH parameters for the threshold variant from [AM<sup>+</sup>23].

### 1.4 Soft Analytical Side-Channel Attacks and Belief Propagation

Soft Analytical Side-Channel Attack (SASCA) was originally proposed by Veyrat-Charvillon et al. [VGS14] with the aim of combining the "divide and conquer" approach and algebraic side-channel attacks, in order to recover keys using sidechannel leakages. Combining these two approaches, we can benefit from the low time and memory complexity and high noise tolerance of the divide-and-conquer approach, as well as the optimal data complexity of algebraic attacks. A real advantage of SASCA over algebraic SCAs, which uses "hard" information, is its ability to manipulate probability distributions, typically those present at the output of classifiers. Indeed, to perform this type of attack, it is necessary to identify a good set of operations dependent on secret input data. Within this set of operations, we will be particularly interested in the intermediate values that are calculated. By using the outputs of classifiers, we obtain probability distributions for these values. We then want to enhance our prediction of the secret by modeling a Maximum a Posteriori Problem that considers the gathered distributions. Typically, we can benefit from the BP algorithm [Pea82] on a factor graph [KFL01], similarly to [VGS14].

The BP algorithm was first introduced by Pearl et al. [Pea82]. It is a message passing algorithm designed to calculate the marginalisation of a function from its factorisation. In the context of SASCA, we can model the relations between intermediate values using a graph. As described above, we can feed this graph with probability distributions obtained with SCA. On this type of graph, also known as a factor graph, we can use the BP algorithm. Specifically, in this factor graph, the probability distributions of the intermediate values are stored in variable nodes and the links between these values are stored in factor nodes. To initialise the algorithm, we give the variable nodes a former "belief" marginal obtained with an SCA, such as the output of a classifier. For variable nodes with no prior knowledge, we initialise them with a uniform distribution. Then, in order to calculate the marginal distributions, the algorithm performs iterative exchanges of messages between the nodes.

More specifically, there are two types of message we can use, each with its own update rule. Thus, using the notations of [Mac03], we can define the two rules as follows:

#### From variable to factor:

$$q_{n \to m}(x_n) = \prod_{m' \in \mathcal{M}(n) \setminus m} r_{m' \to n}(x_n),$$

where  $\mathcal{M}(n)$  denotes the set of factors in which n participates.

#### From factor to variable:

$$r_{m \to n}(x_n) = \sum_{\mathbf{x}_m \setminus n} \left( f_m(\mathbf{x}_m) \prod_{n' \in \mathcal{N} \setminus n} q_{n' \to m}(x_{n'}) \right),$$

where  $\mathcal{N}(n)$  denotes the indices of the variables that the *m*-th factor depends on and  $\mathbf{x}_{m \setminus n}$  denotes the set of variables in  $\mathbf{x}_m$  with  $x_n$  excluded.

Finally, we can get the marginal distributions of all the variables. Specifically, the marginal function of  $x_n$  is obtained by multiplying all the incoming messages

at that node:

$$Z_n(x_n) = \prod_{m \in \mathcal{M}(n)} r_{m \to n}(x_n)$$

We can then normalise these marginals distributions. To do so, we use the normalising constant Z, that we can obtain by summing any marginal function,  $Z = \sum_{x_n} Z_n(x_n)$ , and we compute:

$$P_n(x_n) = \frac{Z_n(x_n)}{Z}.$$

If the factor graph is tree-like, then the above algorithm returns the exact marginals. Unfortunately, in practice, factor graphs often contain cycles, in which case the algorithm does not necessarily converge. To overcome this problem, the loopy-BP algorithm has been proposed, which generally gives sufficiently accurate approximations of the marginals. Although the exact conditions leading to convergence of the loopy-BP algorithm are not known, Su et al. [SW15] have listed some interesting workarounds, such as message damping, that can be useful depending on the context.

### 2 Point of Vulnerability and Construction of the Attack

In this section, we show how to construct an attack on the TCitH framework in order to recover the secret key and we apply it to the SDitH algorithm. To do this, we start by highlighting a side-channel vulnerability of this framework, which we then adapt to SDitH.

#### 2.1 Point of Vulnerability

In order to mount an attack on the TCitH framework, we need the following requirements to be fulfilled:

- 1. Firstly, the search of a function manipulating the secret key. We will concentrate here on the functions used in the MPC protocol, i.e., executed by the parties and which manipulate a share of the secret key or part of the secret key of each party. This part therefore depends on the implementation of the algorithm targeted by the attack.
- 2. In a second phase, we are going to take a closer look at the TCitH framework itself. Specifically to the construction of the shares given as input of the MPC protocol for the parties. This will enable us to make a link between the shares of the secret key and the secret key in order to reconstruct it or part of it. This part is independent of the implementation of the algorithm.

We can then see how to adapt this attack to SDitH:

- 1. We start by targeting the Galois field multiplication function  $gf_mul$  in the polynomial evaluation function. Specifically, we choose to target the *S* polynomial as we can quickly find the secret key from it. The  $gf_mul$  function has the desired characteristics, i.e. being used in the MPC protocol and manipulating the share  $[s_A]$  of each party.
- 2. We use the equation (1), that link  $[\![s_A]\!]_i$  to  $s_A$ , in order to reconstruct  $s_A$  in the secret key.

### 2.2 Exploitation of the Vulnerability with SASCA

We can represent the vulnerability of SDitH introduced in the previous section in the form of a factor graph. One advantage of this representation is that we can also represent the construction part, which is independent of the implementation. This representation allows us to perform a SASCA, which is even more interesting for this algorithm as it presents redundancy, due to the MPC protocol that we repeat  $\tau$  times. Since this redundancy is favourable to SASCA, this enables us to have better results when we retrieve the secret key.

Now that we have seen how to exploit the vulnerability in SDitH, we will see in the next sections how to perform our attack both in simulation and in practice.

# 3 The Attack - Simulation Part

This section presents the theoretical part of our attack on SDitH. It consists of the template on the gf\_mul function, the construction of the SASCA graph and some attack simulations.

### 3.1 The Template

As we said in the previous section, we want to conduct our template attack on a function in the MPC protocol that manipulates a share of a variable linked to the private key, i.e., using the notation introduced earlier, we are looking for a function that manipulates a share  $[\![w]\!]_i$  of the witness, or a part of this witness.

In the case of SDitH, this function is the evaluation in  $r_k$  of a polynomial. And as we said earlier, this evaluation is composed of the gf\_mul function, so we will execute our attack on gf\_mul.

#### 3.1.1 Attacker Model

The type of attack we are considering in this article requires an attacker capable of performing profiling attacks on the SDitH signature, in order to recover the secret key. Our attacker therefore needs a clone of the physical device on which we are going to perform the attack. It is from this clone that he will be able to carry out the profiling phase. For ease of use, we have chosen to carry out the profiling and attack procedures on the same physical device. In the profiling phase, the attacker must be able to create models, in this case from the gf\_mul operation.

To create these models, it is assumed that the attacker can isolate a sequence of gf\_mul calculations within a larger sequence, such as the execution of the MPC protocol by the parties. This part of model creation will not be studied in this paper. We will assume that it can be achieved using matching techniques.

### 3.1.2 Experimental Setup

To acquire our traces, we chose the STM32F407 as our device under test, as it is a commonly used board in the state of the art [KFT<sup>+</sup>21], [GMGL24]. More precisely, we acquired our traces using a "Langer Near Field" electromagnetic probe connected to a Rhode-Schwarz RT02024 oscilloscope with a sampling frequency of 1 GHz. As previously mentioned, we are interested in the gf\_mul multiplication function of the SDitH reference implementation. This code is compiled with the -O3 optimisation, surrounded by a GPIO-based trigger, and allows us to collect traces of 1125 points each for an execution time of  $42\mu s$ .

### 3.1.3 Templates on Galois Field Multiplication

In the following, tests will be performed using the following reference implementation from  $[AM^+23]$  for the gf\_mul function:

Algorithm 1 Multiplication in  $GF(2^8)$  from [AM<sup>+</sup>23]

```
#define MODULUS 0x1B
1
2
  uint8_t gf256_mul(uint8_t a, uint8_t b) {
3
      uint8_t r;
4
      r = (-(b))7
                      ) & a);
      r = (-(b>>6 \& 1) \& a) (-(r>>7) \& MODULUS)
                                                        (r+r);
6
        = (-(b>>5 \& 1) \& a) (-(r>>7) \& MODULUS)
                                                        (r+r);
      r
        = (-(b>>4 \& 1) \& a) (-(r>>7) \& MODULUS)
      r
                                                        (r+r);
8
        = (-(b>>3 & 1) & a) ^ (-(r>>7) & MODULUS)
      r
                                                        (r+r);
9
          (-(b>>2 & 1) & a) ^
                                (-(r>>7) \& MODULUS)
                                                        (r+r);
      r
        = (-(b>>1 \& 1) \& a) (-(r>>7) \& MODULUS)
      r
                                                        (r+r):
11
   return (-(b
                   & 1) & a) ^
                                (-(r>>7) \& MODULUS)
                                                        (r+r):
12
13 }
```

The gf\_mul function is a "schoolbook" style implementation, which means it is more likely to leak. So to check that the gf\_mul function is leaking, we are going to use the Linear Regression Analysis (LRA) as a leakage assessment tool. LRA is a statistical method whose aim is to predict the value of a variable, called the dependent variable, from the value of other variables, called the independent variables. To do this, we build a model in which we express the leakage for a side-channel measurement (in this case, a measurement of one byte of gf\_mul) in linear form. Here we used the linear model defined in [GMGL24]. Furthermore, as the LRA is univariate, it must be applied for each time sample.

Once this model has been built, we want to measure its performance, that is, how well the model fits the data and how well it can predict future results. To do this, we will use the  $R^2$  metric, also known as the coefficient of determination. This will allow us to calculate the proportion of the total variation in the dependent variable that is captured by the linear model based on the independent variables. The closer  $R^2$  is to 1, the more accurate the model is. So we compute the coefficient of determination for the different variables involved in gf\_mul, i.e. the two operands and the output, which gives us the **Figure 2**:

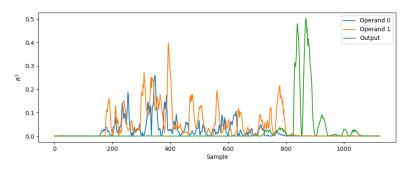


Fig. 2: Coefficients of determination computed for both inputs and the output of the Galois field multiplication

Thanks to **Figure 2**, we can make several observations:

- the leakage of the operand 0 is distributed throughout the calculation, but is not very important;
- the leakage of the operand 1 is distributed in the same way as the first, but is more important, and this due to the numerous shifts performed on this operand;
- the output of the gf\_mul calculation leaks at the end of the function, when it is stored in the main memory, and its leakage is greater than the one of the operands. This can be explained by the numerous logical operations performed on this output.

Now that the analysis of the coefficient of determination has shown us that the gf\_mul function leaks, we can build our templates. More precisely, we build three templates on the gf\_mul function that targets the HW of its inputs (operands 0 and 1) and its output. We start by training these templates with 100000 training traces and by splitting the datasets into 90% training traces and 10% validation traces. After that, we use Fisher's Linear Discriminant Analysis (LDA) to project our data into a lower-dimensional space and as our classifier. Moreover, we trained each LDA on the HW of one of the three variables of interest.

For the LDA, we tested different values for the size of the dimensional space. This shows that the accuracy value is higher when the data is projected into a 2- or 3-dimensional space. In order to have a better visual representation, we therefore chose to use 2 dimensions for **Figure 3**. As we consider our data in a 2-dimensional space, we can then get a first idea of the separability of the different classes with the following **Figure 3**:

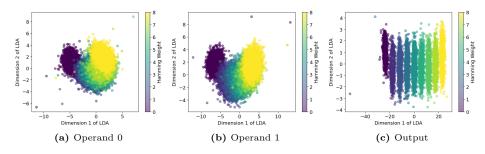


Fig. 3: LDA projection of the traces

We can also calculate the accuracy of our models. For this we repeat each attack 100 times and we summarised them in **Table 3**.

Table 5. Hamming weight accuracy on grimu.						
	Hamming weight template accuracy					
Operand 0	0.4023					
Operand 1	0.5142					
Output	0.9965					

Table 3: Hamming weight accuracy on gf\_mul.

**Discussion.** We can see that the HW of the output can be found almost all the time, with an accuracy of 99.65%. For the two operands, we also have satisfactory results, with respectively 40.23% and 51.42% accuracy for operands 0 and 1.

In the same way as for the coefficient of determination, the number of logical operations performed on the variables of interest has a direct effect on the percentage of accuracy obtained. This can be seen visually in the separability of the HW classes in **Figure 3**.

Even if we have an accuracy of 99.65% for the output, we must not forget that it is in HW, therefore it does not mean that 99.65% of  $s_A$ 's coefficients can be found. Indeed, if we consider that the information we have is the HW of the output and the value of the operand 1 (because the public matrix r\_powers

is behind this one), since the function gf\_mul is non-injective, so there are still many possible values for the operand 0.

As we said earlier, an advantage of a SASCA attack is that using the HW templates on multiple intermediate variables, we can exploit the links between them to recover the values of the coefficients of  $s_A$ .

### 3.2 SASCA Graph and Simulations

In this section, we build a factor graph that models the polynomial evaluation for the SDitH algorithm from which we will be able to perform simulations of SASCA with the HW leaker.

#### 3.2.1 Construction of the Graph

For SDitH, we illustrate our graph in **Figure 4**. Specifically, it is first composed of the evaluation of the polynomial S at points  $r_k$  by the different parties. The second part of the graph is built from the structure of the threshold variant, more precisely using the construction of parties' shares below:

input\_share[e][i] = input\_plain + 
$$\sum_{j=1}^{l} f_i^j \cdot \text{input_coef}[e][j]$$
 when  $i \neq N$ . (2)

We recall that this construction is independent from the implementation of SDitH and comes directly from the mathematical specificities of this algorithm. This will enable us to make the link between input\_coef  $(S_0, S_1 \text{ and } S_2)$  and input\_plain (S). Specifically, we will retrieve the first k bytes of S, corresponding to the value of  $s_A$ . It is from this value that we will be able to reconstruct the Q and P polynomials that make up the rest of the secret key.

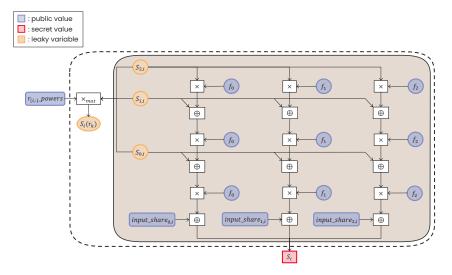
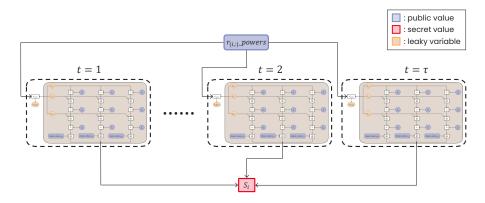


Fig. 4: Evaluation of polynomial S factor graph.

In **Figure 4**, we have three types of factors: (i) the XOR factor  $\oplus$ , (ii) the gf\_mul factor  $\times$ , and (iii) and the vector/matrix multiplication factor  $\times_{mat}$  composed of the gf\_mul factor. In multiplication  $\times_{mat}$ , r\_powers is the matrix composed of the powers of the random points  $\{r_k\}_{k \in [t]}$  with  $r_k$  sampled from  $\mathbb{F}_{points} = \mathbb{F}_{q^n}$ . We can also see on this figure the brown block that corresponds to the reconstruction part of S from equation (2) and that is independent from the implementation of SDitH. The dotted block will be used later to describe the entire graph, i.e. repeated several times.

We repeat this graph for each byte  $i, i \in [0:k]$ , to recover the secret key  $s_A$ . Moreover, when the signature algorithm is run, the MPC protocol is performed  $\tau$  times. In other words, we repeat  $\tau$  times our full graph (graph repeat for all bytes) with other values for  $S_j, f_j$  and input\_share i  $(j \in [0, 2])$ .

Note that the variable  $r_powers$  remains the same for all  $\tau$ . Moreover, each time the graph is run, the reconstruction part of the S polynomial is performed in the same S variable. For a coefficient of S we then have the graph **Figure 5**:



**Fig. 5:** Full factor graph for a coefficient of S

At the end, we have the above graph for the k coefficients of S. Since this full graph contains cycles, it benefits from the loopy-BP convergence proof.

#### 3.2.2 Simulated Attack

Now that we've built our graph, let us simulate our attack using a noisy HW model. These simulations give us a generic analysis of our attack.

As we can see if we look at the reference implementation and at the **Figure 4**, the leaky variables are used as the first operands and the output in the function  $gf_mul$ . So we start by initialising the marginal probabilities of the first operand and the outputs for all  $gf_mul$  computation from a HW leakage model with a Gaussian noise  $\sigma$ .

Moreover, we can define the leakage model that is affected by the manipulation of a  $gf_mul$  operand 0, denoted a, with the following formula:

$$L(a) = \alpha \cdot \mathrm{HW}(a) + \beta,$$

with  $\beta \sim \mathcal{N}(0, \sigma)$  and  $\alpha = 1$  for an easier interpretability of the results. We can define the leakage model in the same way for an output of gf\_mul.

**Remaining Complexity.** When we run the simulations, we notice that for the coefficients of the S polynomial multiplied by the first row of the  $r_powers$  matrix, the BP algorithm does not converge to the correct coefficients. This can be explained by the particular form of this first line, which is made up exclusively of 0 and 1s, unlike the other lines. The nature of the information available to us (HW on the output) and the absence of "useful" redundancy, i.e., multiplication by 0 and 1 only, means that, even with perfect knowledge of the HW, it is not possible to isolate a single candidate for these coefficients of S. As SDitH uses a split version of the SD problem, there are exactly d coefficients (including the first coefficient) of the polynomial S which are multiplied by the first line, so the convergence problem affects d coefficients (i.e. d = 1 for security level I and d = 2 for security levels III and V).

We can see, by looking at how the syndrome x is constructed in the key generation algorithm, that one byte of x has a  $\frac{m-w}{m}\%$  (about 65% for all security levels) chance of being 0. And so, by constructing S, its first coefficient also has the same percentage chance of being 0. When it is not, since we are using simulated HW leakage, we can find this coefficient in at most 70 attempts, i.e how many elemnts in GF(256) have a HW of 4 (as this is the most represented HW). When d = 2, for the second coefficient that does not converge, we will also need 70 attempts. The bruteforce complexity is therefore of 70 for security level I and of  $70^2$  for security levels III and V.

So in the following, we will assume that the attack has succeeded when we find the k-d coefficients of the polynomial S. Once our attack simulations have been performed, we illustrate them with **Figure 6**, which shows the success rate of the simulated attack as a function of the noise parameter  $\sigma$ . Each attack was simulated 50 times for each security level of SDitH.

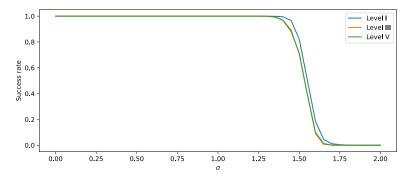


Fig. 6: Simulated success rate of SASCA on the polynomial evaluation of SDitH.

**Simulation results.** We observe that we can find all the coefficients for  $\sigma \leq 1.4$  for security level I, and for  $\sigma \leq 1.3$  for security levels III and V.

The curves also show that security levels III and V are very close. This can be explained by the fact that there are 62 more S coefficients to be found between levels III and V, whereas in comparison there were more (94) between levels I and III. On the other hand, between each level the value of  $\tau$  increases by 3, and m by 16. And during the simulations we noticed that the value of  $\tau$  had a non-negligible impact on the number of coefficients retrieved, i.e. more coefficients can be found for the same noise level. So, considering these two remarks, it can explain why the curves for safety levels III and V are this close.

# 4 The Attack - Practical Part

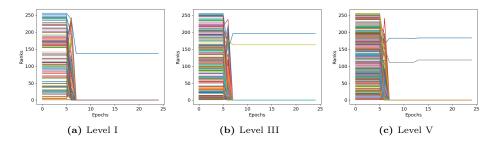
In order to robustly compute success rates on a practical attack, we use the structure of prediction matrices. Prediction matrices have been introduced in [GMGL24] as a structure which aims at providing a repeatable "real-case-like" simulations by storing multiple template predictions. Specifically, when we randomly sampled elements from such a matrix, we can see it as a real template prediction. To construct these prediction matrices, we associate each value of the coefficients of S with an estimate of the leakage model based on HW (computed thanks to the template predictions). Precisely, we construct a dictionary that contains the coefficients of S as keys and array of output distributions as values.

And in the same way as we did for the HW leaker, we build our prediction matrices on the first operand and the output. Therefore, three attacks can be derived from this:

- 1. leakage on the operand 0 and the output;
- 2. leakage on the operand 0;
- 3. leakage on the output.

**Results.** Now that we have constructed the prediction matrices, we are able to execute our attacks. For the first and third attacks, they show a perfect success rate for all the security levels on SDitH, and can be performed in a few minutes. For the second one, due to the accuracy of the operand 0 in **Table 3**, the BP graph does not converge and the attack does not succeed as might be expected.

In what follows, we define the term "epochs" as the iterations during which the BP algorithms operate on our graph factor. Each epoch represents a cycle of information updates between nodes, progressively refining the attacker's knowledge of the secret key. In order to get an idea of the efficiency of these attacks, we will look at how the different coefficients of  $s_A$  converge depending on the number of epochs of the BP algorithm in **Figure 7** and **Figure 8**:



**Fig. 7:** Evolution of the rank of  $s_A$  coefficients as a function of the number of epochs using real HW leakages on operand 0 and output.

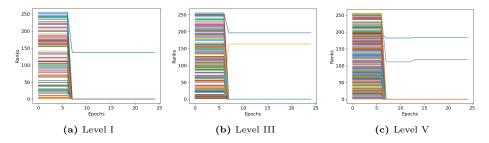


Fig. 8: Evolution of the rank of  $s_A$  coefficients as a function of the number of epochs using real HW leakage on the output.

If we look at the **Figure 7** and **Figure 8**, for which the attack succeeds, we can see that we need 7 iterations of the BP algorithm to achieve such a result. We can also see, as we explained in the SASCA simulations, that d coefficients of S do not converge on the correct value in our graph.

### 5 Shuffling Countermeasures

As mentioned above, SASCA attacks have already targeted Kyber's NTT. To counter these attacks, Ravi et al. proposed in [RPBC20] two shuffling countermeasures: fine shuffling and coarse shuffling. The security of these countermeasures has been studied (and shown to be insufficient) in [GMGL24] for HQC and adapted to vector/matrix multiplication using the gf\_mul operation. Therefore, in the same way we can adapt them for SDitH.

**Fine shuffling.** The fine shuffling can be used to randomise the input of  $gf_{mul}$ , allowing to manipulate the secret data (i.e., a coefficient of S) behind operand 0 once in two cases. To understand more precisely how this countermeasure works, we can use the following pseudo-code of **Algorithm 2**:

**Algorithm 2** Fine shuffling for a function f with two inputs

**Require:** f a function, a and b two inputs **Ensure:** f(a,b) or f(b,a)  $r \stackrel{\$}{\leftarrow} \{0,1\}$  **if** r = 0 **then return** f(a,b) **else return** f(b,a)**end if** 

**Coarse shuffling.** The coarse shuffling itself can shuffle the rows/columns together in the r\_powers matrix. This type of shuffling changes the sequence in which operations, i.e.  $gf_mul$ , in  $\times_{mat}$  are performed. As with fine shuffling, we can explain coarse shuffling in more detail using the pseudo-code of **Algorithm 3** below:

Algorithm 3	Coarse	shuffling	for	a fur	nction	of	multi	plication	vector	/matrix	
-------------	--------	-----------	-----	-------	--------	----	-------	-----------	--------	---------	--

However, following the same reasoning as for HQC, these two countermeasures are not effective for SDitH.

[GMGL24] also introduced new countermeasures, called window shuffling and full shuffling.

Window shuffling. For the window shuffling, the idea is to compute each window in a random order, allowing us to change the order of the coefficients in the computation of S. We consider here an iteration of our graph in **Figure 4** as a window, and as they are independent, we can use this countermeasure. Changing the order of execution of the windows leads the BP algorithm to converge on a permutation of the real value of S. We can also explain this countermeasure with the pseudo-code in **Algorithm 4**:

Algorithm 4 Window shuffling

Require: f_window	
$index\_window = random\_perm$	$nutation([0, 1, \ldots, k-1])$
for $i$ in index_window <b>do</b>	
$\operatorname{Execute}(f\_window(i))$	$\triangleright$ a window corresponds to the graph in Figure 4
end for	

Once again, as for HQC and using the Hungarian algorithm, this countermeasure is not effective for SDitH.

Full shuffling. The idea for full shuffling is to combine the window shuffling and the coarse shuffling. This result in the randomisation of the order of gf\_mul calculations and leads to an overhead equal to the cost of shuffling a list the size of the r\_powers matrix, i.e.,  $(m/d) \times r_{cols}$ . And as we have that the size of the r\_powers matrix is much larger than the size of the Galois field, as in [GMGL24], the complexity of full shuffling inversion corresponds to the number of permutations of the r\_powers matrix, meaning that we respectively have a complexity of  $2^{1208}$ ,  $2^{1248}$  and  $2^{1449}$  for the three security levels of SDitH. Consequently, this countermeasure is efficient against the attack proposed in this paper.

# 6 Conclusion and Further Work

In this paper, we presented the first side-channel attack on the TCitH framework, specifically the first single trace attack, targeting the secret key. We begin by highlighting a point of vulnerability which we can divide into two parts and that we exploit on SDitH:

- the first part depends on the implementation of the chosen algorithm (here SDitH) in which we are looking for a function run by the parties in the MPC protocol and that manipulates a share of the secret. This function corresponds to the polynomial evaluation in SDitH.
- the second part is independent of the implementation and creates a link between the shares of the secret and the actual secret, i.e., the coefficients of  $s_A$  for SDitH.

Furthermore, we exploit the redundancy inherent to this type of framework through SASCA.

After a leakage assessment showing leakages on the Galois field multiplication, we built templates so as to predict the Hamming weight of the operands and output. We then use SASCA to directly retrieve the values of  $s_A$ . We simulated the leakages with increasing noise level in order to evaluate the noise resilience of SASCA on SDitH. We were able to recover the full secret key (to within dcoefficients) for  $\sigma \leq 1.4$  for security level I and for  $\sigma \leq 1.3$  for security levels III and V. With real physical measurements, we were able to perform attacks with either (i) leakage on the operand 0 and the output, (ii) leakage on operand 0, and (iii) leakage of the output. Only the second attack does not work, as we might have expected given the value of the accuracy previously obtained. For the two other attacks, we can retrieve the full private key with an bruteforce complexity of 70 for security level I, and of  $70^2$  for security levels III and V. In addition, we can run them in just a few minutes.

We end this paper with a discussion on some shuffling countermeasures. This enabled us to determine that the full shuffling [GMGL24] represents a good countermeasure for the SDitH algorithm.

Future work could investigate to which extent the structure of our attack can be adapted to the new TCitH [FR23] and VOLEitH [BBdSG<sup>+</sup>23] frameworks, and if so, how. Furthermore, as signature schemes can allow an attacker to gather traces of several signatures, we believe that multi-trace attack scenarios, profiled or not, are an interesting research path.

# Acknowledgement

This work was supported by the French National Research Agency in the framework of the "Investissements d'avenir" program (IRT Nanoelec, ANR-10-AIRT-05).

### References

- AM<sup>+</sup>23. C Aguilar-Melchor et al. The syndrome decoding in the head (SD-inthe-Head) signature scheme. submission to the NIST call for additional post-quantum signatures, 2023.
- BBdSG<sup>+</sup>23. Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Emmanuela Orsini, Lawrence Roy, and Peter Scholl. Publicly verifiable zero-knowledge and post-quantum signatures from vole-inthe-head. In Helena Handschuh and Anna Lysyanskaya, editors, Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part V, volume 14085 of Lecture Notes in Computer Science, pages 581–615. Springer, 2023.
- BFR23. Ryad Benadjila, Thibauld Feneuil, and Matthieu Rivain. MQ on my mind: Post-quantum signatures from the non-structured multivariate quadratic problem. IACR Cryptol. ePrint Arch., page 1719, 2023.
- BMG<sup>+</sup>24. Chloé Baïsse, Antoine Moran, Guillaume Goy, Julien Maillard, Nicolas Aragon, Philippe Gaborit, Maxime Lecomte, and Antoine Loiseau. Secret and shared keys recovery on hamming quasi-cyclic with SASCA. IACR Cryptol. ePrint Arch., page 440, 2024.
- BN20. Carsten Baum and Ariel Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, Public-Key Cryptography PKC 2020 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4-7, 2020, Proceedings, Part I, volume 12110 of Lecture Notes in Computer Science, pages 495–526. Springer, 2020.

- CRR02. Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers, volume 2523 of Lecture Notes in Computer Science, pages 13-28. Springer, 2002.
- FJR22. Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. Syndrome decoding in the head: Shorter signatures from zero-knowledge proofs. In Yevgeniy Dodis and Thomas Shrimpton, editors, Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part II, volume 13508 of Lecture Notes in Computer Science, pages 541–572. Springer, 2022.
- FR22. Thibauld Feneuil and Matthieu Rivain. Threshold linear secret sharing to the rescue of mpc-in-the-head. *IACR Cryptol. ePrint Arch.*, page 1407, 2022.
- FR23. Thibauld Feneuil and Matthieu Rivain. Threshold computation in the head: Improved framework for post-quantum signatures and zeroknowledge arguments. *IACR Cryptol. ePrint Arch.*, page 1573, 2023.
- FS86. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings, volume 263 of Lecture Notes in Computer Science, pages 186–194. Springer, 1986.
- GMGL24. Guillaume Goy, Julien Maillard, Philippe Gaborit, and Antoine Loiseau. Single trace HQC shared key recovery with SASCA. IACR Trans. Cryptogr. Hardw. Embed. Syst., 2024(2):64–87, 2024.
- GMO16. Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster Zero-Knowledge for boolean circuits. In 25th usenix security symposium (usenix security 16), pages 1069–1083, 2016.
- HHP<sup>+</sup>21. Mike Hamburg, Julius Hermelink, Robert Primas, Simona Samardjiska, Thomas Schamberger, Silvan Streit, Emanuele Strieder, and Christine van Vredendaal. Chosen ciphertext k-trace attacks on masked CCA2 secure kyber. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):88– 113, 2021.
- IKOS07. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zeroknowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007, pages 21–30. ACM, 2007.
- KFL01. Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. Inf. Theory*, 47(2):498–519, 2001.
- KFT<sup>+</sup>21. Tendayi Kamucheka, Michael Fahr, Tristen Teague, Alexander Nelson, David Andrews, and Miaoqing Huang. Power-based side channel attack analysis on PQC algorithms. *IACR Cryptol. ePrint Arch.*, page 1021, 2021.
- Mac03. David J. C. MacKay. Information theory, inference, and learning algorithms. Cambridge University Press, 2003.

- MGH<sup>+</sup>22. Carlos Aguilar Melchor, Nicolas Gama, James Howe, Andreas Hülsing, David Joseph, and Dongze Yue. The return of the sdith. *IACR Cryptol. ePrint Arch.*, page 1645, 2022.
- NIS16. CFP NIST. Submission requirements and evaluation criteria for the postquantum cryptography standardization process, 2016.
- Pea82. Judea Pearl. Reverend bayes on inference engines: A distributed hierarchical approach. In David L. Waltz, editor, Proceedings of the National Conference on Artificial Intelligence, Pittsburgh, PA, USA, August 18-20, 1982, pages 133–136. AAAI Press, 1982.
- PP19. Peter Pessl and Robert Primas. More practical single-trace attacks on the number theoretic transform. In Peter Schwabe and Nicolas Thériault, editors, Progress in Cryptology - LATINCRYPT 2019 - 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2-4, 2019, Proceedings, volume 11774 of Lecture Notes in Computer Science, pages 130–149. Springer, 2019.
- PPM17. Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace sidechannel attacks on masked lattice-based encryption. In Wieland Fischer and Naofumi Homma, editors, Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings, volume 10529 of Lecture Notes in Computer Science, pages 513–533. Springer, 2017.
- RL23. Gina M Raimondo and Laurie E Locascio. Module-lattice-based keyencapsulation mechanism standard. *National Institute of Standards and Technology, Gaithersburg*, 2023.
- RPBC20. Prasanna Ravi, Romain Poussier, Shivam Bhasin, and Anupam Chattopadhyay. On configurable SCA countermeasures against single trace attacks for the NTT - A performance evaluation study over kyber and dilithium on the ARM cortex-m4. In Lejla Batina, Stjepan Picek, and Mainack Mondal, editors, Security, Privacy, and Applied Cryptography Engineering - 10th International Conference, SPACE 2020, Kolkata, India, December 17-21, 2020, Proceedings, volume 12586 of Lecture Notes in Computer Science, pages 123–146. Springer, 2020.
- SW15. Qinliang Su and Yik-Chung Wu. On convergence conditions of gaussian belief propagation. *IEEE Trans. Signal Process.*, 63(5):1144–1155, 2015.
  VGS14. Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In Palash Sarkar and Tetsu Iwata, editors, Advances in Cryptology ASIACRYPT 2014 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I, volume 8873 of Lecture Notes in Computer Science, pages 282–296. Springer, 2014.