Simulating SASCA on Keccak: Security Implications for Post-Quantum Cryptographic Schemes

Julien Maillard[1,2][a], Thomas Hiscock[1][b], Maxime Lecomte[1][c] and Christophe Clavier[2][d]
[1] *Univ. Grenoble Alpes, CEA-LETI, Minatec Campus, f-38054 Grenoble, France*
[2] *Univ. Limoges, XLIM-MATHIS, Limoges, France*

Keywords: Keccak, Side-Channel Attacks, SASCA, Kyber, Dilithium.

Abstract: Keccak is a standard hashing algorithm that is used in cryptographic protocols as Pseudo Random Functions (PRF), as Pseudo Random Number Generator (PRNG), to check data integrity or to create a Hash-based Message Authentication Code (HMAC). In many cryptographic constructions, secret data is processed with hashing functions. In these cases, recovering the input given to the hashing algorithm allows retrieving secret data. In this paper, we investigate the application of Soft Analytical Side-Channel Attacks (SASCA), based on a Belief Propagation (BP) framework, to recover the input of SHA-3 instances. Thanks to a simulation framework, we extend existing work on the Keccak-f permutation function by developing a comprehensive study of the attacker's recovery capacity depending on the hash function variant. Then, we study the security implications of SASCA on cryptosystems performing multiple calls to hashing functions with inputs derived from the same secret data. We show that such constructions can be exploited efficiently by an attacker and show typical use-cases by targeting Kyber's encryption routine and Dilithium's signing routine. We also show that increasing Kyber's security parameters implies weaker security against SASCA. Finally, our study gives insights about the minimal bit-level classification accuracy required for successful SASCA on Keccak.

1 INTRODUCTION

Hashing algorithms are deterministic one-way cryptographic functions that take as input a message of variable size and produce a fix-sized output called a hash or digest. Main use-cases of hashing functions include password storage, integrity verification and *Message Authentication Code* (MAC) creation. Many applications rely on hashing functions such as security protocols (TLS, IPsec, SNMP, *etc.*), blockchain technologies (Bitcoin, Ethereum, *etc.*) and even *Post-Quantum Cryptography* (PQC) schemes , for example Kyber (Avanzi et al., 2019), Dilithium (Lyubashevsky et al., 2020) or SPHINCS$^+$ (Bernstein et al., 2019).

SHA-3 has been standardized by the *National Institute of Standards and Technologies* (NIST) in 2015. It is based on Keccak, a sponge structure built over a permutation function called Keccak-f. When a hashing function manipulates a secret input, an attacker can try to recover the latter by observing side-channel leakages during the hashing process, such as power consumption or electromagnetic radiations. Interestingly, Keccak is used to manipulate secrets in several PQC schemes with different aims. Indeed, it is used as a *pseudo-random function* (PRF) in *Key-Encapsulation Mechanisms* (KEM) such as Kyber and FrodoKEM, or as a *Pseudo-Random Number Generator* (PRNG) in the post-quantum Fiat-Shamir-type signature Dilithium. Finally, instances of post-quantum hash-based signature schemes like SPHINCS$^+$ or XMSS rely on Keccak. These uses of Keccak mainly process either ephemeral or fixed secrets, hence with few input variability. This prevents the application of *Differential Power Analysis* (DPA) approaches that were possible in MAC scenarios (Zohner et al., 2012; Bilgin et al., 2014). To tackle this shortcoming, recent single-trace *Soft Analytical Side-Channel Attacks* (SASCA) targeting the Keccak-f cryptographic permutation function have been exposed in the literature and shown to be practical (Kannwischer et al., 2020; You and Kuhn, 2021), enabling the recovery of these fixed or ephemeral secrets. In (Kannwischer et al., 2020), a list of the algo-

[a] https://orcid.org/0009-0002-2267-7621
[b] https://orcid.org/0009-0001-5183-2291
[c] https://orcid.org/0000-0002-9985-7586
[d] https://orcid.org/0000-0002-0767-3684

Maillard, J., Hiscock, T., Lecomte, M. and Clavier, C.
Simulating SASCA on Keccak: Security Implications for Post-Quantum Cryptographic Schemes.
DOI: 10.5220/0012787200003767
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 21st International Conference on Security and Cryptography (SECRYPT 2024)*, pages 518-527
ISBN: 978-989-758-709-2; ISSN: 2184-7711
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

rithms that could be targeted by SASCA on Keccak-f is provided, including several PQC targets. They also exposed countermeasures against single-trace attacks. They suggest, without proving it, that multiple calls to Keccak, with an input composed of a fixed value concatenated to a counter, can be exploited in lattice-based KEM and digital signatures. In this paper, we revisit these intuitions by mounting a shared-key recovery attack on Kyber, as well as a private key recovery on Dilithium. Later on, You and Kuhn (You and Kuhn, 2021) templated fragments of secret variables, and they exposed high classification accuracies at a bit-level scale on a Chipwhisperer-Lite board(O'flynn and Chen, 2014). They built a bit-level factor graph on several SHA-3 instances and mount successful attacks on this board. Nevertheless, their study did not give insights on the resistance to noise of a bit-level model. In this paper, we perform a simulated SASCA approach at bit-level that aims at evaluating the resistance of single or multiple calls to SHA-3 against SASCA considering bit-level templating.

1.1 Contributions

In this paper, we implement several soft analytical side-channel attacks based on belief propagation theory to recover the input of Keccak-f by leveraging the side-channel leakage of inter-round states. We extend the work presented in the literature through a comprehensive study of the resistance of standard SHA-3 instances regarding SASCA in a simulated context. Namely, we investigate the accuracy of SASCA with an increasing noise level rather than on a particular device. We then investigate how SASCA can merge information when multiple hash functions are called with inputs derived from a same secret through the analysis of SHAKE-256 based error vector derivation in Kyber post-quantum cryptography standard. Our quantitative approach allows to better comprehend that the use such “multiple calls” structures present an additional security concern. Finally, we discuss the implications of a similar attack scenario on a Dilithium signature to recover the private key.

2 BACKGROUND

2.1 SASCA and Belief Propagation

A side-channel attacker that targets a cryptographic application is able to gain probabilistic information regarding the value of several intermediate variables. We assume that the attacker knows the cryptographic application they are attacking. Thus, they know the mathematical relationships that link all intermediate variables in the algorithm. The idea behind SASCA is to combine likelihoods gathered from side-channel analysis in order to derive a *Maximum A Posteriori* (MAP) estimation of the marginal distribution of a secret. This can be performed by modeling the link between intermediate variables within a bipartite graphical model called a factor graph. This model allows dividing the high dimensional problem of marginal estimation into a set of smaller dimensional problems. A factor graph contains two types of nodes. Firstly, variable nodes are used to store the probability distributions of the target algorithm's intermediate variables. Secondly, factor nodes represent the arithmetical links between two or more variables. Upon this graph, the MAP estimation is carried out by a message passing algorithm, called belief propagation, where likelihoods are transmitted between variable nodes and factor nodes. The message $\mu_{x \to g}$ sent from variable node x to factor node g is defined as follows:

$$\mu_{x \to g}(x) = \prod_{h \in n(x) \setminus \{g\}} \mu_{h \to x}(x) \tag{1}$$

where $n(x)$ corresponds to the set of neighboring nodes of x (*i.e.*, connected to x with an edge) in the factor graph. Additionally, messages sent by a factor g to a variable x is computed with the *sum-product* formula depicted as follows:

$$\mu_{g \to x}(x) = \sum_{\sim\{x\}} \left(f(X) \prod_{y \in n(g) \setminus \{x\}} \mu_{y \to f}(y) \right) \tag{2}$$

where X represents the set of variable nodes connected to g and $\sim\{x\}$ expresses the summary notation as defined in (Kschischang et al., 2001). Note that f is a boolean function representing the arithmetical link between variables in $n(g)$. Finally, the marginal distribution of a variable node is computed as follows:

$$P(x) = \frac{1}{Z} \prod_{g \in n(x)} \mu_{g \to x}(x) \tag{3}$$

with a normalization factor Z. Factor graphs representing cryptographic functions are often cyclic. Hence, an iterative message passing algorithm, called loopy-BP, is applied until convergence.

2.2 Keccak Specifications

SHA-3 is a hashing algorithm based on the Keccak-f permutation function (Dworkin, 2015). Keccak-f takes as input a 5×5 matrix of elements of size 2^l-bits, $l \in \{3,4,5,6\}$, that are processed through five sub-routines θ, ρ, π, χ and ι, that are called sequentially for $12 + 2l$ rounds. Within the SHA-3 framework, Keccak-f calls are organized with a sponge construction. The sponge construction consists in two phases: "absorption", where the data is injected within the primitive, and "squeezing" where the hash is provided to the user. Keccak-f input of size b bits is divided into two parts of respective sizes: the "rate" r and the "capacity" $c = b - r$. The security level against collision and preimage attacks is $\frac{c}{2}$. The input of SHA-3, denoted P is padded and then divided into chunks of size r, denoted $\{P_0, ..., P_{n-1}\}$, that are absorbed one by one by Keccak-f. Then, the squeezing part delivers chunks $\{Z_0, ..., Z_{t-1}\}$ of size r that are concatenated to build the final hash. In this paper, we only focus on SHA-3 versions with $l = 6$, with 64-bit words and 24 rounds (this version is called Keccak[1600]). Depending on the desired application, one can use Keccak through one of the standard instances depicted in Table 1.

Table 1: Parameters of standard SHA-3 instances.

Algorithms	r	c
SHAKE-128	1344	256
SHA3-224	1152	448
SHA3-256, SHAKE-256	1088	512
SHA3-384	832	768
SHA3-512	576	1024

3 RELATED WORK

Soft analytical side-channel attacks have been introduced by Veyrat *et al.* on an AES Furious implementation (Veyrat-Charvillon et al., 2014). Interestingly, SASCA was shown to outperform *Algebraic Side-Channel Attacks* (ASCA), even in noise-free contexts (Grosso and Standaert, 2015). Grosso *et al.* showed that SASCA required much less training traces than profiled DPA attacks. Later on, SASCA was adapted to key recovery on Kyber by targeting the number theoretic transform (Primas et al., 2017; Pessl and Primas, 2019; Hamburg et al., 2021; Hermelink et al., 2023). Kannwischer *et al.* mounted the first single trace SASCA on Keccak (Kannwischer et al., 2020). Their approach used clustered nodes in order to shift the representation of variables from chunk-level (*i.e.*, 8-bit or 16-bit) to bit-level in order to model the θ transform of Keccak-f. They also provide insights about the repercussions of SASCA on Keccak upon post-quantum schemes that are based on the latter. Later, You and Kuhn developed a fully bit-level approach of SASCA on Keccak (You and Kuhn, 2021). Thanks to bit-level likelihoods gathered from a fragment template attack, they targeted a Keccak-f[1600] implementation on a Cortex-M4 device.

4 ATTACKER MODEL

In this paper, we consider a profiled attack scenario where an adversary can train a model, or template, on a clone of the target board that runs the exact same algorithm, but with controlled data. The adversary is also able to perform leakage assessment (*e.g.*, to identify the leakage model or to select a set of points of interest for templating) on any intermediate variables of the target algorithm. We consider the attacker to be able to craft templates for inter-round intermediate variables only (including the hashing function input). This choice provides interesting insights regarding the security of hardware implementations of hashing functions. Indeed, several hardware implementations of SHA-3 instances use registers to store inputs and outputs of permutation rounds for Keccak-f (Arshad et al., 2014; Sundal and Chaves, 2017; Michail et al., 2015). Note that, in practice, a classifier that targets an inter-round variable may exploit intra-round leakages, this is especially the case for complex models (*e.g.*, deep neural networks). By default, we restrict the attack phase to the observation of a single side-channel trace. This model can be relaxed when, for example, hash computations are performed as part of secure boot implementations. Since the hashing can be replayed with data when rebooting the device, better prediction accuracies could be obtained.

4.1 Prior Likelihoods Generation

As mentioned in section 2, Keccak[1600] handles 64-bit variables. Although the templating of 32-bit variables has been shown possible (Cassiers et al., 2023), manipulating probability distributions upon 2^{32} possible values requires a high amount of storage, and this becomes impractical for 64-bit variables. An alternative strategy called fragment template approach, consists in dividing a n-bit variable into smaller chunks (or fragments), and applying a template attack on each chunk. Interestingly, You and Kuhn (You and Kuhn, 2021) showed that the fragment size has no significant impact on bit-level marginal distribution prediction. This means that, in their particular case, a bit-level fragment template attack is as relevant as for other fragment sizes. As Keccak-f uses bitwise manipulations, a factor graph at bit-level scale is appropriate. Consequently, we consider an arbitrary bit-level classifier to conduct our experiments. In order to evaluate the correction capacity brought by SASCA depending on the performance of such bit-level classifier, we introduce the notion of virtual leakage.

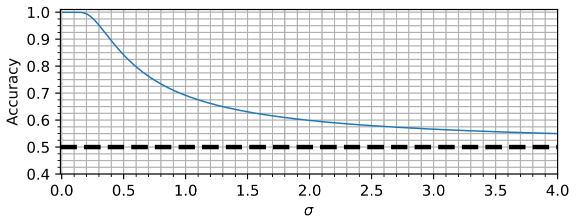

Figure 1: Accuracy of a bit-level classifier for increasing σ, the standard deviation of the noise within the latent space after dimensionality reduction. The black dashed line corresponds to a random guess.

Virtual Leakage. The leakage of an intermediate variable is supposed multivariate (*i.e.*, spread over multiple time samples). Then, we consider an arbitrary bit-level classifier that relies on the projection the side-channel measurements into a latent space (which can be viewed as dimensionality reduction). We propose an abstraction of such dimensionality reduction by directly parametrizing the data distribution, *i.e.,* the virtual leakage, in this latent space. While, in a real case scenario, the data distribution in this latent space depends on multiple factors (*e.g.*, leakage models, learning algorithm), we opt for a simulation with few parameters. Let X be the random variable representing bit values, and L the random variable representing the leakage. For each bit x, the virtual leakage l is defined as follows:

$$l = x + \beta,\ \beta \sim \mathcal{N}(0, \sigma^2) \tag{4}$$

Noise and Accuracy. Template matching is used on drawn samples to generate conditional probabilities $P(X = x \mid L)$. As we consider two classes, each one standardized, the accuracy of template matching can be computed as follows:

$$Acc = 1 - P(L \leq 0.5 | X = 1) \tag{5}$$

with $P(L \leq l)$ being the cumulative density function. Accuracy depending on noise parameter of the virtual leakage σ is depicted in Figure 1. As expected, when σ increases, the accuracy converges towards a random guess. This procedure allows to alter the accuracy of a bit-level classifier, emulating the impact of measurement noise.

Key Ranking. We evaluate the accuracy of the attacks in this paper thanks to key rank estimation (Veyrat-Charvillon et al., 2013; Grosso, 2019; Poussier et al., 2016). Such procedure provides an estimation of the remaining complexity of a key enumeration strategy after an attack. Hence, the lower the rank, the quicker an attacker would find the correct key with an appropriate key enumeration algorithm. We consider all individual bits of the secret (*i.e.*, contained in the "rate" part of Keccak-f's input) as subkeys. The likelihoods obtained on these bits after SASCA can be provided to the key rank estimation algorithm. We rely on the algorithm presented in (Poussier et al., 2016), which is based on histogram convolutions to combine likelihoods of the subkeys. The main parameter to setup in this method is the number of bins defining the histogram. The fewer bins, the faster the execution algorithm but the higher the estimation error. In this study, the size of the secret is often superior to 512 bits, so we choose a rather small number of bins (*i.e.*, 50 bins) but we average key rank estimations results over several attack runs, so as to minimize the impact of estimation error. Finally, the key rank metric is represented with logarithmic scale within this work.

Baseline Attack. To provide a comparison basis, we introduce the notion of "baseline attack" which describes an attack that is only headed thanks to templating the input variables (without BP). For example, the accuracy of a baseline attack on a hashing function is obtained by only templating its input. The baseline allows to evaluate the correction capacity brought by SASCA. We define the correction capacity as the percentage of accuracy gained with SASCA. For example, a successful SASCA with prior likelihoods originating from a bit-level classifier with an accuracy of 0.9 corresponds to a 10% correction capacity.

Limitations. In this work, we consider the latent space of the dimensionality reduction phase of an arbitrary bit-level classifier to follow a normal distribution. This assumption is based on the central limit theorem. Indeed, we suppose that if the target intermediate variable leaks at several time instants and with enough different leakage models in side channel-measurements, a projection could result in normally distributed data in a latent space. However, this is highly dependent on the nature of the projection and the parameters of the initial leakage. Still, this model provides insights upon minimal accuracy levels required for a bit-level classifier for a successful attack.

5 KECCAK-F FACTOR GRAPH

In this paper, we build a bit-level factor graph based on the work of You and Kuhn (You and Kuhn, 2021). Namely, each one of the 1600 bits of the state is represented thanks to a variable node, this for each round of Keccak-f. The subgraph representing the θ routine incorporates intermediate variable nodes representing the parity bits. Note that π and ρ routines are implemented with a simple wiring. The factor graph contains three different factor types which respectively represent the bitwise "exclusive-or", "and" and "not" operations (this latter one can easily be implemented with a lookup table). The exclusive-or operation is performed thanks to a Walsh-Hadamard transform (Kannwischer et al., 2020). The per-class likelihoods outputted from an arbitrary classifier on an intermediate variable can be incorporated within a factor-graph thanks to an observational factor. This factors' only function is to transmit these likelihoods to the variable node it is connected to. In our simulated context, we restrict observational factors to inter-rounds intermediate variables only (*i.e.*, excluding θ, and χ intermediate variables). This differs from the approach taken in (You and Kuhn, 2021), where θ intermediate variables corresponding to parity were connected to observational nodes, allowing explicit intra-round leakage exploitation. As mentioned in section 4, this allows to gain intuition upon the resistance of hardware implementations against SASCA. Nevertheless, one can insert observational nodes inside the rounds, and we expect that this would increase attack performances. Finally, when not stated otherwise, we consider by default the whole rate bits to constitute the secret (see Table 1).

6 SINGLE CALL ATTACKS

The aim of this section is to evaluate a general attack scenario where a side-channel adversary has the possibility to apply previously crafted templates upon a measurement of a single call to Keccak. Along this section, we aim at addressing the following questions: *(i)* What is the minimal bit-level template accuracy achievable for a successful attack? *(ii)* What is the minimum number of Keccak-f rounds to profile to obtain satisfactory recovery? *(iii)* What accuracies can be expected for different instances of Keccak?

Fully Unknown Input. The case of a fully unknown Keccak-f input state of 1600 bits represents a late invocation of Keccak-f in the sponge construction, without prior knowledge of the output of the previous invocation. In (You and Kuhn, 2021), authors report a success rate close to 0 on the recovery of a state with more than 1500 unknown bits. Consequently, they need to know at least a part of the capacity, which implies a successful attack on prior invocations. We believe that such dependency between the attacks can be detrimental to an attacker, so we also investigate SASCA on fully random Keccak-f inputs.

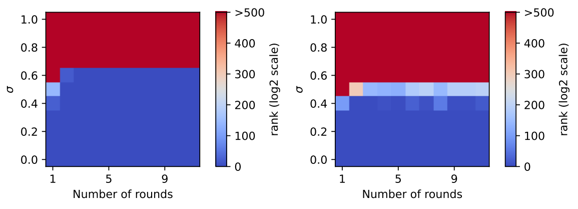

Figure 2: Average key rank estimation on the input after 50 runs of simulated BP attacks on Keccak-f[1600] with full unknown inputs (right) and $r = 1088$ unknown bits, the SHA3-256 FIPS standard (left).

6.1 Number of Target Rounds

Firstly, the purpose of our analysis is to assess the minimal number of rounds that must be profiled by an attacker in order to get satisfactory recovery of the input considering noise. Indeed, this reduces the number of variables to be templated by the attacker, with hopefully similar recovery potential. We perform two simulations with varying noise parameter σ and number of templated rounds. The first experiment evaluates the average key rank estimation for the input of a SHA3-256 / SHAKE-256 instance (with rate $r = 1088$). The second experiment measures the average key rank of a fully unknown Keccak-f input state of 1600 bits. In Figure 2 we observe that attacking SHA3-256 allows the good key to be ranked first up to $\sigma = 0.6$, which is better than for a full random input. Still, with $\sigma < 0.5$, the rank of the good guess stays under 2^{69}, which is approximately the number of blocks hashed by the bitcoin network in one second in January 2024 (Coinwarz,). We also notice that, regardless the targeted version, no significant improvement is brought by profiling more than 5 Keccak-f rounds (2 rounds for SHA3-256). We believe that this is related to the non-injective nature of the bitwise logical-and operation performed in χ, which limits the backwards propagation of information within the graph. Further Keccak-f related analyses in this paper will be conducted considering templates on 5 rounds. Note that this is slightly higher than (Kannwischer et al., 2020) and (You and Kuhn, 2021), that used respectively 2 and 3 (or 4, depending on the version of SHA-3) rounds.

6.2 Evaluation of SHA-3 Attacks

We now investigate the resistance of standard SHA-3 instances (see Table 1) against SASCA. For an increasing noise level, we run the attack 50 times on random inputs and computed key ranks are averaged. In order to assess the benefits of SASCA against a baseline attack, the result of a baseline classifier, as defined in section 4, is used as a reference. Results of

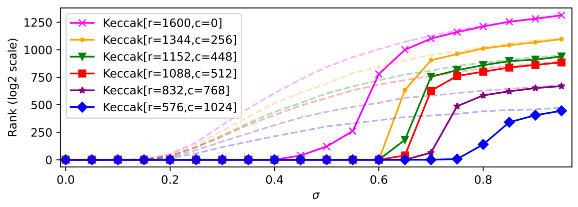

Figure 3: Comparison of the key ranks of a baseline classifier (without BP, light dashed lines) and SASCA attacks on Keccak-f for several rate r and capacity c parameters.

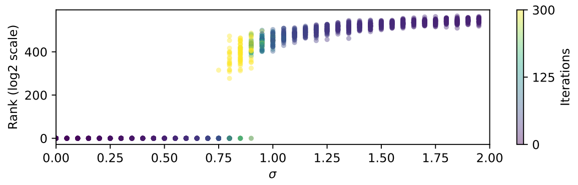

Figure 4: Number of iterations of the loopy-BP algorithm for the SHA3-512 attack. The maximum number of iterations is set to 300.

such simulations are illustrated in Figure 3.

Results provided in Figure 3 show that the proportion of known data plays an important role regarding the resistance of the attack to noise. Indeed, for Keccak instances, the average rank after the attack is expected to converge up to the size of the problem (*i.e.*, 2^r). Also, the greater the capacity size, the greater amount of noise can be supported by SASCA leading to a rank of 0. This follows observations made in (You and Kuhn, 2021) and (Kannwischer et al., 2020).

Benefits of SASCA. In Figure 3, we observe that the good guess rank after baseline attacks increases rapidly after a bit-level noise value of $\sigma = 0.15$, whereas the least efficient SASCA (*i.e.*, with full unknown input state, depicted by Keccak[$r = 1600$, $c = 0$]) exposes an average rank of 0 up to an initial bit-level classifier accuracy of 0.9 (corresponding to $\sigma = 0.4$ in Figure 1). With the increasing noise level, we observe the average ranks after both baseline and SASCA approaches converge towards an random guess. Consequently, we can conclude that, in the cases considered in this study, SASCA is always beneficial to an attacker within a noise domain depending on the capacity of SHA-3. Average key rank after SASCA converges more slowly towards random guessing when the noise increase, being beneficial to the adversary when key enumeration is possible.

Noise and BP Convergence. In Figure 4, we illustrate the ranks and the number of loopy-BP iterations obtained for several runs of a SHA3-512 attack for an increasing noise level. A threshold on the maximal statistical change of marginals from one iteration to the next is setup as a stopping criterion for loopy-BP. For low noise levels (*i.e.*, inferior to $\sigma = 0.70$), the number of iterations for all runs stays low (*i.e.*, inferior to 70) while the rank of the good guess is 0. This means that the loopy-BP algorithm quickly converges to the good hypothesis. From $\sigma = 0.75$ to $\sigma = 0.9$, the intermediate zone, the majority of runs that do not lead to low rank reach the maximum number of iterations (300 in this experiment), indicating that loopy-BP did not converge. Finally, after $\sigma = 0.90$, the number of iterations for all the runs drops below 70, while the rank of the good hypothesis remains high. In this case, the loopy-BP algorithm converges, but on a wrong hypothesis. If, in a black box context, an attacker observes that loopy-BP algorithm reaches the maximum number of iterations, this means that the initial template accuracy probably corresponds to the intermediate zone depicted in Figure 4. The attacker, then, could adjust the classification model, or retry the attack with new measurements.

6.3 Discussion

We presented a methodology to assess the minimal number of hashing function rounds necessary to mount successful SASCA on SHA-3. We analyzed the benefits of SASCA compared to a baseline attack, and confirmed the statement provided in (You and Kuhn, 2021) and (Kannwischer et al., 2020): the more input is known, the higher is the resistance of SASCA to noise. This implies that Keccak instances with smaller rate r, while having higher security against cryptanalysis, are more sensitive to SASCA.

Additionally, the intermediate key transition zone in Figure 4, where most of the loopy-BP instances saturate at the maximum number of iterations, can be explained by the potential appearance of oscillations or error floors. This could reveal trapping or absorbing sets within the factor graph (Dolecek et al., 2009). Moreover, no significant improvement was brought by increasing the maximal number of iterations (*i.e.*, loopy-BP still frequently hits the maximum in this intermediate domain), neither by applying message damping (this was used in (Kannwischer et al., 2020) and shown inefficient in a bit-level factor graph in (You and Kuhn, 2021)).

Future work could aim at characterizing absorbing sets in such noise domains, in order to opt for an appropriate way to counter them. Specifically, adding weights to the message passing process with finer granularity than classical message damping could be beneficial. Hence, we believe that neural enhanced belief-propagation models (Satorras and Welling, 2021) should be considered in further works.

7 MULTIPLE CALLS ATTACK

In (Kannwischer et al., 2020), authors briefly mention that several calls to Keccak-f with an input composed of a fixed part (a random coin here) and a varying part (a counter value in this case) could be exploited by merging the factor graphs in order to aggregate information, following an approach depicted in (Veyrat-Charvillon et al., 2014). In this section, we study the vulnerability of such structures regarding SASCA. For this sake, we take as use-cases a shared key recovery on Kyber, and a private key recovery on Dilithum. Note that both these attacks suppose that the attacker is able to detect and label different calls to the hashing function. In practice, this brings additional technical difficulties that should be taken into account.

7.1 Kyber's Shared Key Recovery

Kyber is a post-quantum *Key Encapsulation Mechanism* (KEM) that has been standardized by the NIST in 2022 (Avanzi et al., 2019). Kyber is a lattice-based scheme that aims at encapsulating a shared key for secure key exchange. It relies on an encryption procedure that consists in projecting the message (*i.e.*, the shared key) in a lattice and adding an error, which is derived from a secret coin. To reach IND-CCA2 security, the receiver needs to perform a Fujisaki-Okamoto transform (Fujisaki and Okamoto, 1999), that includes a re-encrypt operation. This means that a side-channel attack targeting the encryption function to recover the shared key can either target the sender or the receiver device.

Encryption Function. Pseudocode of the encryption algorithm used in Kyber is depicted in Algorithm 1.This algorithm takes as input a secret message. As one can see, a 32-byte secret random coin r is derived thanks to a PRF in order to generate a secret vector $\hat{\mathbf{r}}$ and error vectors $\mathbf{e_1}$ and e_2. We stress that, with the knowledge of the ciphertext c (which can be intercepted by the adversary), the public key pk and the secret random coin r, an attacker is able to recover the message m.

Attacking the PRF. The derivation in Kyber uses SHAKE-256 as a PRF. As seen in Table 1, SHAKE-256 is based on Keccak[r=1088, c=512]. Namely, in Algorithm 1., r is manipulated with $PRF(r,N)$, for $N \in \{0,\dots,2k\}$, with $PRF(r,N) =$ SHAKE-256$(r||N)$. As r is a 256-bit variable and N is known at each step, it is possible to mount an attack targeting the $N_{tot} = 2k+1$ calls to SHAKE-256.

```
Input: Public key $pk \in \mathcal{B}^{12\cdot k\cdot n/8+32}$
Input: Message $m \in \mathcal{B}^{32}$
Input: Random coin $r \in \mathcal{B}^{32}$
Result: Ciphertext $c \in \mathcal{B}^{d_u\cdot k\cdot n/8+d_v\cdot n/8}$
$N \leftarrow 0$
$\hat{\mathbf{t}} \leftarrow \mathtt{decode}_{12}(pk)$
$\hat{\mathbf{A}} \leftarrow \mathtt{generate\_public\_matrix}(pk)$
for $i$ from $0$ to $k-1$ do
    $\mathbf{r}[i] \leftarrow CBD_{\eta_1}(PRF(r,N))$
    $N \leftarrow N+1$
end
for $i$ from $0$ to $k-1$ do
    $\mathbf{e_1}[i] \leftarrow CBD_{\eta_2}(PRF(r,N))$
    $N \leftarrow N+1$
end
$e_2 \leftarrow CBD_{\eta_2}(PRF(r,N))$
$\hat{\mathbf{r}} = NTT(\mathbf{r})$
$\mathbf{u} = NTT^{-1}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \mathbf{e_1}$
$v = NTT^{-1}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + e_2 + \mathtt{decompress}_q(\mathtt{decode}_1(m), 1)$
$c_1 = \mathtt{encode}_{d_u}(\mathtt{compress}_q(\mathbf{u}, d_u))$
$c_2 = \mathtt{encode}_{d_v}(\mathtt{compress}_q(v, d_v))$
return $c = (c_1||c_2)$
```

Algorithm 1: Kyber encryption function (Avanzi et al., 2019).

Note that the higher N_{tot}, the higher is Kyber's security level. By looking backwards at Algorithm 1, we see that c_2 can be recovered from c, which is a simple concatenation of c_1 and c_2. Then, an attacker can compute v' as follows:

$$v' = \mathtt{decompress}_q(\mathtt{decode}_{d_v}(c_2), d_v) \tag{6}$$

We know from Kyber's specification that the loss of information induced by this computation is:

$$\delta_{err} = |v - v' mod^{\pm} q| \leq \left\lceil \frac{q}{2^{d_v+1}} \right\rfloor \tag{7}$$

with $\lceil x \rfloor$ being the rounding operation. The value of δ_{err} corresponding to each security level of Kyber is depicted in Table 2.

By setting $m_{dec} = \mathtt{decompress}_q(\mathtt{decode}_1(m), 1)$, from Algorithm 1we have:

$$v = NTT^{-1}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + e_2 + m_{dec} \tag{8}$$

By making the hypothesis that an attacker is able to recover r, they can then obtain $NTT^{-1}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + e_2$. Then, the attacker can compute the following:

$$m'_{dec} = v' - \left(NTT^{-1}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + e_2\right) \tag{9}$$

This operation guarantees that $|m_{dec} - m'_{dec} mod^{\pm} q| = \delta_{err}$. Then we have:

$$\mathtt{compress}_q(m'_{dec}, 1) = \left\lceil \frac{2}{q} \cdot m_{dec} \right\rfloor \mathrm{mod}^{+} 2 \tag{10}$$

For all Kyber security levels, the relation $\delta_{err} < \frac{q}{4}$ is guaranteed. This allows the attacker to compute the shared key m with:

$$m = \mathtt{encode}_1(\mathtt{compress}_q(m'_{dec}, 1)) \tag{11}$$

Table 2: Kyber parameters values.

Version	q	k	N_{tot}	d_v	δerr
Kyber512	3329	2	5	4	104
Kyber768	3329	3	7	4	104
Kyber1024	3329	4	9	5	52

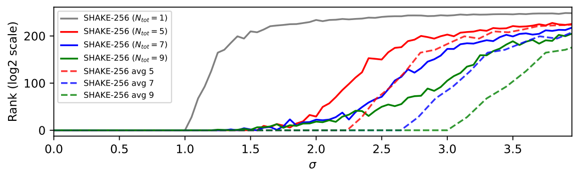

Figure 5: Average rank 50 runs with increasing noise on Kyber's encryption attack considering multiple SHAKE-256 calls. Dotted lines represent the trace averaging version of the attack with $N_{tot}=1$.

Simulation Results. We create a joint factor graph that represents intermediate variables of all N_{tot} SHAKE-256 calls, built around the bit-level variables that represent the 256-bit secret coin r. Kannwischer et al. expected that attacks in this setting would perform similarly than where the input is not changing and where trace averaging is possible (Kannwischer et al., 2020). Let X be a random variable of mean μ_x and standard deviation σ_x. When drawing random samples from X of size n, the central limit theorem for sample means states that, considering the random variable $\bar{X}$ consisting in sample means, we have:

$$\bar{X} \sim N\left(\mu_x, \frac{\sigma_x}{\sqrt{n}}\right) \tag{12}$$

Our simulated model allows to easily estimate the average rank of an attack considering fixed input, with trace averaging taken into account. In this case, we simply need to apply a factor $\sqrt{N_{tot}}$ to the noise supported by the attack with $N_{tot}=1$ to simulate the "trace averaging" scenario.

Attack results are illustrated in Figure 5. First, one can observe that the noise level supported by the attack of a single SHAKE-256 call is superior to what is depicted in Figure 3. Indeed, even if SHAKE-256 is based on Keccak[r=1088, c=512], the secret coin r is concatenated to a known value N and then padded with the Keccak standard padding scheme. This leads to 1344 known bits and 256 unknown bits within the latter input. The size of the manipulated secret being smaller than all previously analyzed Keccak instances, the observed accuracy for the SHAKE-256 ($N_{tot}=1$) instance is higher than for instances in Table 1. This follows the conclusions brought in section 6. Most importantly, considering several calls to SHAKE-256 sharing the same secret allows to drastically increase the level of noise supported by the attack, following the evolution of N_{tot}. Furthermore, we bring a nuance to the statement of (Kannwischer et al., 2020). Indeed, SASCA results on merged graphs is less resistant to noise than SASCA on the averaged fixed input scenario. We believe this can be explained by topological aspects of the merged factor graph that could lead to the appearance of absorbing and trapping sets. We can, however, consider the "trace averaging version" as a loose upper bound that can be used to quickly evaluate security. Finally, further work could aim at tightening the gap between approximation by investigating methods to enhance the loopy-BP procedure, as stated in subsection 6.2.

7.2 Dilithium Private Key Recovery

Dilithium (Lyubashevsky et al., 2020) is a lattice based post-quantum Fiat-Shamir digital signature scheme that has been standardized by the NIST. During the randomized signing routine, a secret random token $\rho' \in \{0,1\}^{512}$ is drawn, and processed, concatenated to a counter variable κ through the *ExpandMask* function, which is implemented thanks to SHAKE-256. The recovery of ρ', along with the knowledge of public parameters and one (or few) messages and corresponding signatures, is enough to retrieve the full secret key (Berzati et al., 2023). Due to the rejection sampling process, the signing process makes, in average, 4.25 calls to the *ExpandMask* function. Then, thanks to simple power analysis, an adversary could spot the different calls to *ExpandMask*, and associate the appropriate value of κ for each one of them. Then, an attacker could create a merged graph, similarly to the approach taken in subsection 7.1. Also, the more the rejection sampling fails to output a valid signature, the more calls to *ExpandMask* are performed, which would lead to a wider graph, and a higher resistance of SASCA to noise (see Figure 5). Consequently, conclusions about the vulnerability induced by multiple calls of SHAKE-256 with a derivation of the same secret also hold for Dilithium's signing routine.

8 SUMMARY

We display the minimal bit-level template accuracy required for successful SASCA attack on average in Table 3. Firstly, we observe that, except for the case of Kyber related graphs, the same bit-level classification accuracy is needed to reach either a 2^{68} or 2^{0} rank. This is due to a quick drop in SASCA correction capacity. Still, for SHA3-512, the correction capacity of SASCA reaches approximately 23.5%. This goes higher when aggregating graphs in the case of Kyber, where the correction capacity is of 37.5%. Fi-

Table 3: Minimal Required bit-level fragment template accuracies for successful SASCA attack.

Algorithms	rank 2^0	rank 2^{69}
Keccak-f$[r = 1600, c = 0]$	0.875	0.875
SHAKE-128	0.8	0.8
SHA3-224	0.8	0.8
SHA3-256, SHAKE-256	0.775	0.775
SHA3-384	0.765	0.765
SHA3-512	0.765	0.75
Kyber graph ($N_{tot} = 1$)	0.69	0.675
Kyber graph ($N_{tot} = 5$)	0.625	0.6
Kyber graph ($N_{tot} = 7$)	0.625	0.58
Kyber graph ($N_{tot} = 9$)	0.630	0.57

nally, with an enumeration effort roughly equivalent to a second of bitcoin network workload in January 2024 (*i.e.*, 2^{69}) (Coinwarz,), the correction brought by SASCA and enumeration can reach more than 40% in the best case scenario (*i.e.*, Kyber graph with $N_{tot} = 9$).

9 COUNTERMEASURES

All attacks presented in this paper rely on the ability of an attacker to craft a template attack on several intermediate variables within the targeted hashing functions. Hence, masking measures would mitigate these attacks (Groß et al., 2017; Arribas et al., 2018). We showed that SASCA does not benefit from templating variables after the fifth permutation round. Hence, masking only the first Keccak-f rounds would allow reducing the amount of necessary randomness while providing satisfactory security guarantees, this with reduced computational overhead. All attacks presented in this paper are valid until a certain bit-level prediction accuracy. Hence, classical measures that tend to limit the power consumption variations or that insert dummy cycles during the sensitive code execution can be beneficial. Furthermore, as formulated in (Kannwischer et al., 2020), a shuffling countermeasure consisting in reordering operations within Keccak-f routines would complicate the template construction phase of the attack, and thus lower the SASCA accuracy. Indeed there are very few data dependencies within Keccak-f routines, allowing to easily sample random permutations.

Whereas aforementioned measures come with a cost in terms of required randomness and execution time, countermeasures can be taken at protocol level. Indeed, in section 7, we show that multiple derivations of a same secret going through a hash function raise a vulnerability. Moreover, in the special case of Kyber, we see that increasing the security level directly enhances the potential for an attacker to mount attacks that support a higher level of noise. Consequently, possible countermeasure paths for future schemes could be headed towards implementing derivations from a secret that are harder to exploit from an attacker's perspective, typically by preventing an adversary to easily aggregate several factor graphs. This could be done without any additional necessary randomness by, for example, leveraging the "squeeze" phase of Keccak. However, theoretical security of such methods remains to be evaluated.

10 CONCLUSION

In this paper, we investigated the security of SHA-3 hashing functions against soft analytical side-channel attacks that aim at retrieving a secret input under a bit-level leakage model. Thanks to a simulated approach, we assess the threat represented by SASCA on most common SHA-3 version, thus extending previous works on that topic. Next, we mount an attack that exploits multiple calls to a hashing function that processes data derived from the same secret. We evaluate this approach on Kyber's encryption function and Dilithium signing routine, that are based on SHAKE-256. This type of construction leads to additional vulnerabilities when considering an attacker with SASCA potential. Particularly, it shows that attacks based on multiple hash function calls really push forward the acceptable noise level from an attacker's perspective. In cases where profiled side-channel attacks are possible, the attacks described in this paper represent a threat. This must be taken into account when designing new schemes that use hashing function to manipulate secret data, particularly when multiple deviations from the same secret are needed.

In this paper, we deliberately eluded the practical difficulties of performing a template attack on hashing functions, this to focus on an exhaustive study of the noise resistance of SASCA. Hence, future works could investigate such practical aspects, and the impact of considering within SASCA likelihoods coming from models that profile variables with different leakage models. This could lead to the application of advanced profiling, for example based on deep neural networks, allowing to investigate the interface between machine learning and probabilistic graphical models within a SASCA framework.

ACKNOWLEDGEMENT

This work has benefited from a government grant managed by the National Research Agency under France 2030 with reference ANR-22-PECY-0009.

REFERENCES

Arribas, V., Bilgin, B., Petrides, G., Nikova, S., and Rijmen, V. (2018). Rhythmic Keccak: SCA security and low latency in HW. *IACR Transactions on Cryptographic Hardware and Embedded Systems*.

Arshad, A., Aziz, A., et al. (2014). Compact implementation of SHA3-512 on FPGA. In *Conference on Information Assurance and Cyber Security (CIACS)*.

Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., and Stehlé, D. (2019). Crystals-kyber algorithm specifications and supporting documentation. *NIST PQC Round*.

Bernstein, D. J., Hülsing, A., Kölbl, S., Niederhagen, R., Rijneveld, J., and Schwabe, P. (2019). The sphincs+ signature framework. In *Proceedings of the 2019 ACM SIGSAC*.

Berzati, A., Viera, A. C., Chartouny, M., Madec, S., Vergnaud, D., and Vigilant, D. (2023). Exploiting intermediate value leakage in dilithium: a template-based approach. *IACR Transactions on Cryptographic Hardware and Embedded Systems*.

Bilgin, B., Daemen, J., Nikov, V., Nikova, S., Rijmen, V., and Van Assche, G. (2014). Efficient and first-order dpa resistant implementations of keccak. In *Smart Card Research and Advanced Applications, CARDIS 2013*.

Cassiers, G., Devillez, H., Standaert, F. o.-X., and Udvarhelyi, B. (2023). Efficient Regression-Based Linear Discriminant Analysis for Side-Channel Security Evaluations: Towards Analytical Attacks against 32-bit Implementations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*.

Coinwarz. Bitcoin hash rate. https://www.coinwarz.com.

Dolecek, L., Zhang, Z., Anantharam, V., Wainwright, M. J., and Nikolic, B. (2009). Analysis of absorbing sets and fully absorbing sets of array-based ldpc codes. *IEEE Transactions on Information Theory*.

Dworkin, M. J. (2015). SHA-3 standard: Permutation-based hash and extendable-output functions.

Fujisaki, E. and Okamoto, T. (1999). Secure integration of asymmetric and symmetric encryption schemes. In *Annual international cryptology conference*.

Groß, H., Schaffenrath, D., and Mangard, S. (2017). Higher-order side-channel protected implementations of keccak. In *2017 Euromicro Conference on Digital System Design (DSD)*.

Grosso, V. (2019). Scalable key rank estimation (and key enumeration) algorithm for large keys. In *Smart Card Research and Advanced Applications, CARDIS 2018*.

Grosso, V. and Standaert, F.-X. (2015). ASCA, SASCA and DPA with enumeration: which one beats the other and when? In *Advances in Cryptology, ASIACRYPT 2015*.

Hamburg, M., Hermelink, J., Primas, R., Samardjiska, S., Schamberger, T., Streit, S., Strieder, E., and van Vredendaal, C. (2021). Chosen ciphertext k-trace attacks on masked cca2 secure kyber. *IACR Transactions on Cryptographic Hardware and Embedded Systems*.

Hermelink, J., Streit, S., Strieder, E., and Thieme, K. (2023). Adapting belief propagation to counter shuffling of NTTs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*.

Kannwischer, M. J., Pessl, P., and Primas, R. (2020). Single-trace attacks on keccak. *Cryptology ePrint Archive*.

Kschischang, F. R., Frey, B. J., and Loeliger, H.-A. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*.

Lyubashevsky, V., Ducas, L., Kiltz, E., Lepoint, T., Schwabe, P., Seiler, G., Stehlé, D., and Bai, S. (2020). Crystals-dilithium. *Algorithm Specifications and Supporting Documentation*.

Michail, H. E., Ioannou, L., and Voyiatzis, A. G. (2015). Pipelined SHA-3 implementations on FPGA: Architecture and performance analysis. In *Proceedings of the Second Workshop on Cryptography and Security in Computing Systems*.

O'flynn, C. and Chen, Z. (2014). Chipwhisperer: An open-source platform for hardware embedded security research. In *Constructive Side-Channel Analysis and Secure Design: 5th International Workshop, COSADE 2014*.

Pessl, P. and Primas, R. (2019). More practical single-trace attacks on the number theoretic transform. In *Progress in Cryptology–LATINCRYPT 2019*.

Poussier, R., Standaert, F.-X., and Grosso, V. (2016). Simple key enumeration (and rank estimation) using histograms: An integrated approach. In *Cryptographic Hardware and Embedded Systems, CHES*.

Primas, R., Pessl, P., and Mangard, S. (2017). Single-trace side-channel attacks on masked lattice-based encryption. In *Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*.

Satorras, V. G. and Welling, M. (2021). Neural enhanced belief propagation on factor graphs. In *International Conference on Artificial Intelligence and Statistics*.

Sundal, M. and Chaves, R. (2017). Efficient FPGA implementation of the SHA-3 hash function. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*.

Veyrat-Charvillon, N., Gérard, B., Renauld, M., and Standaert, F.-X. (2013). An optimal key enumeration algorithm and its application to side-channel attacks. In *Selected Areas in Cryptography, SAC 2012*.

Veyrat-Charvillon, N., Gérard, B., and Standaert, F.-X. (2014). Soft analytical side-channel attacks. In *Advances in Cryptology–ASIACRYPT 2014: 20th International Conference on the Theory and Application of Cryptology and Information Security*.

You, S.-C. and Kuhn, M. G. (2021). Single-trace fragment template attack on a 32-bit implementation of keccak. In *International Conference on Smart Card Research and Advanced Applications*.

Zohner, M., Kasper, M., Stöttinger, M., and Huss, S. A. (2012). Side channel analysis of the sha-3 finalists. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*.