A Study of Soft Analytical Side-Channel Attacks on Secure Hash Algorithms

Julien Maillard[1,2], Thomas Hiscock[1], Maxime Lecomte[1] and Christophe Clavier[2]

[1] Univ. Grenoble Alpes, CEA, Leti, MINATEC Campus, F-38054 Grenoble, France
[2] XLIM, University of Limoges, Limoges

Abstract. Hashing algorithms are one-way functions that are used in cryptographic protocols as Pseudo Random Functions (PRF), to assess 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 two popular hash function families: SHA-2 and SHA-3. Thanks to a simulation framework, we develop a comprehensive study of the attacker's recovery capacity depending on the hash function variant. Then, we demonstrate that an attacker can leverage prior knowledge on the hashing function input to increase the effectiveness of the attacks. As an example, in the context of a bootloader doing a hash-based integrity check on a secret firmware, we show that simple statistics on assembly code injected in BP improves input recovery. Finally, we study the security implications of SASCA on cryptosystems performing multiple invocations of hashing functions with inputs derived from the same secret data. We show that such constructions can be exploited efficiently by an attacker. We support such statements with experiments on SHA-256 based HMAC and on SHAKE-256 based PRF in Kyber's encryption routine. We also show that increasing Kyber's security parameters implies weaker security against the proposed SASCA targeting the shared key.

Keywords: Side-Channel Attack · SASCA · SHA-256 · SHA-3 · HMAC · Kyber

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 technonogies (Bitcoin, Ethereum, *etc.*) and even post-quantum cryptography schemes (Kyber, Dilithium, SPHINCS, *etc.*).

The *Secure Hash Algorithm* (SHA) is a family of hashing algorithms that have been standardized by the *National Institute of Standards and Technologies* (NIST). Even if SHA-3 instances have been standardized in 2015, the SHA-2 family (SHA-256, SHA-512, SHA-224, SHA-384) is still commonly used. At the time of this writing, SHA-256 based *hash-based message authentication code* (HMAC) is still enabled by default in SSH, TLS, or IPsec.

The SHA-3 family differs from SHA-2 in terms of structure. Indeed, the first is based on a sponge construction built over a permutation function called Keccak-f, whereas SHA-2 is based on a Merkle Damgård scheme relying on a compression function. Note that

nowadays, no algebraic attack thwarts the theoretical security of SHA-2, which stays a robust candidate for the aforementioned hashing function use-cases.

When a hashing function manipulates a secret as an 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.

So far, the security of SHA-2 implementations has been studied through *Differential Power Analysis* (DPA) [MTMM07,BBD+13,KGB+18] and *Template Attacks* (TA) [BDT+21]. Concerning the SHA-3 family, recent *Soft Analytical Side-Channel Attacks* (SASCA) targeting the Keccak-f cryptographic permutation function have shown to be practical [KPP20, YK21]. SASCA relies on a probabilistic graph model designed to compute marginal distributions on intermediate variables by combining the output of multiple TAs performed on the same side-channel trace. The framework aims at correcting prediction errors resulting from TAs by exploiting the link between intermediate values, which is known in advance by an attacker.

This paper aims at answering the following questions: *(i) What is the resistance of SHA-2, compared to SHA-3, against SASCA? (ii) Can an attacker exploit the structure of the input of a hashing function in order to enhance SASCA accuracy? And (iii) To which extend multiple calls to a hashing function, with their inputs derived from the same secret, add an additional vulnerability considering SASCA?*

1.1 Contributions

In this paper, we implement several soft analytical side-channel attacks based on belief propagation theory to recover the input of a hashing function by exploiting the leakage of inter-round states. Thanks to simulations, we evaluate the robustness of SHA-256 against SASCA, and provide additional insights, with respect to the literature, on the resistance of SHA-3 to BP-based attacks with a bitwise model. The main contributions of this paper are the following:

- We implement the first BP-based attack on the SHA-256 compression function. We perform simulated attacks to find the minimal number of compression rounds to target with TAs for a successful outcome.

- We extend the work presented in the literature by investigating 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 compare the resistance of SHA-256 and SHA-3 instances against SASCA. We claim that the compressive structure of SHA-256 is beneficial from an attacker's perspective, and that the Keccak-f permutation function is more resistant to the attacks we propose.

- We show how an attacker can leverage prior knowledge on the hashing function input to increase the effectiveness of the attacks. As an illustration, we consider the scenario of firmware authentication with SHA-256, and show that an attacker can exploit simple statistics on the *Instruction Set Architecture* (ISA) of the firmware to obtain higher attack accuracies.

- We then investigate how SASCA can exploit joint information when multiple hash functions are called with inputs derived from the same secret. Through the analysis of SHA-256 based HMAC and SHAKE-256 based error vector derivation in Kyber post-quantum cryptography standard, we show that protocols that use such "multiple invocations" structures present an additional security concern.

All attacks presented in this paper can be performed with less than 30 gigabytes of memory and a few minutes with a parallel implementation of the BP algorithm on a regular desktop computer.

Finally, this work deliberately focuses on simulated attacks only. This approach is led by performing an exhaustive study of SASCA accuracy with varying noise levels. This allows us to provide insights upon the theoretical resistance of SHA-3 and SHA-2 against SASCA: such insights can then be used in an evaluation phase of a particular software or hardware implementation of such hashing functions.

1.2 Outline

Necessary background notions are introduced in Section 2. Previous works from the literature are described in Section 3. We present attacker model considerations in Section 4. In Section 5 we present the construction of factor graphs for SHA-256 and Keccak-f. We mount attacks exploiting single invocations of the hashing function in Section 6. Then, we investigate the benefits of exploiting the underlying structure of input data in Section 7. We present an attack on SHA-256 HMAC construction in Section 8. Afterwards, we exploit multiple calls of SHAKE-256 in order to mount an attack on Kyber's encrypt routine in Section 9. After presenting possible mitigations to our attacks in Section 10 we conclude and provide leads regarding further research in Section 11.

2 Background

2.1 SASCA and Belief Propagation

A side-channel attacker that targets a cryptographic application is often able to gain probabilistic information regarding the value of several intermediate variables. In most cases, 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 the likelihoods, gathered from a side-channel analysis phase, in order to derive a *Maximum A Posteriori* (MAP) estimation of the marginal distribution of a secret.

Such MAP estimation can be performed by modeling the link between intermediate variables within a bipartite graphical model called a factor graph. This model allows to factor 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. Factor and variable nodes are connected with edges that represent the "has-argument" statement.

Upon this graph, the MAP estimation is carried out by a message passing algorithm, called belief propagation, where likelihoods, or beliefs, 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 [KFL01]:

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

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 [KFL01]. Note that the f function is a boolean function that represents 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.

The BP algorithm has been proved to compute exact marginals on tree-like graphs. However, factor graphs that represent cryptographic applications often contain cycles. The loopy-BP algorithm has been created to operate on cyclic factor graphs. The key idea of loopy-BP is to iteratively transmit beliefs from factor nodes to variable nodes, and then from variable nodes to factor nodes. Even if loopy-BP is not proved to compute exact marginals, it leads satisfying empirical results in practice.

As the loopy-BP is an iterative algorithm, one has to select one or several arbitrary criteria to stop the algorithm. A criterion can be a maximum number of iterations (set in advance) or a metric evaluating the convergence of the algorithm, such as a threshold on the statistical change of one or several variables' marginal distributions. This work considers both aforementioned stopping criteria.

2.2 SHA-256 Specifications

SHA-2 is a hashing function which has been developed by the *National Security Agency* (NSA) of the United-States. Together with SHA-512, SHA-256 is a member of the SHA-2 family, that has been standardized in 2002. Even if SHA-2 shares similarities with SHA-1, which suffers security flaws, no serious collision attack has been found on SHA-2. Hence, the latter family is still widely in use in cryptographic protocols.

The SHA-256 compression function takes as input a 512-bit message block M and a 256-bit chaining value V. Note that the first chaining value, which is publicly known, is denoted IV. After a message expansion routine, it iterates the same transformation during 64 rounds (see Algorithm 1). The round is composed of several 32-bit boolean operations. Let's define a, b and c three 32-bit variables, the operations are defined as follows:

$$\begin{aligned}
\Sigma_0(a) &= (a \ggg 2) \oplus (a \ggg 13) \oplus (a \ggg 22) \\
\Sigma_1(a) &= (a \ggg 6) \oplus (a \ggg 11) \oplus (a \ggg 25) \\
Ch(a,b,c) &= (a\&b) \oplus (\neg a\&c) \\
Maj(a,b,c) &= (a\&b) \oplus (a\&c) \oplus (b\&c)
\end{aligned}$$

where & denotes the bitwise logical and operation, $\neg$ denotes the bitwise complement and $a \ggg i$ indicates a bitwise right rotation of a of i bits. The message expansion splits the input M into 32-bit words $(M_0, ..., M_{15})$ and computes the extended message $(W_0, ..., W_{63})$ by using the following operations:

$$\begin{aligned}
\sigma_0(a) &= (a \ggg 7) \oplus (a \ggg 18) \oplus (a \gg 3) \\
\sigma_1(a) &= (a \ggg 17) \oplus (a \ggg 19) \oplus (a \gg 10)
\end{aligned}$$

where $a \gg i$ denotes the bitwise logical shift of a of i bits.

Algorithm 1: SHA-256 compression function.

Data: Data block $M = (M_0, ..., M_{15})$ and chaining value $V = (V_0, ...V_8)$
Result: The chaining value $F(V, M)$

$(W_0, ...W_{15}) \leftarrow (M_0, ..., M_{15})$
for $t = 16$ *to* 63 **do**
  $W_t \leftarrow \sigma_1(W_{t-2}) \oplus W_{t-7} \oplus \sigma_0(W_{t-15}) \oplus W_{t-16}$
$(A, B, C, D, E, F, G, H) \leftarrow (V_0, ..., V_7)$
for $t = 1$ *to* 64 **do**
  $T_1 \leftarrow H \oplus \Sigma_1(E) \oplus Ch(E, F, G) \oplus K_t \oplus W_t$
  $T_2 \leftarrow \Sigma_0(A) \oplus Maj(A, B, C)$
  $(H, G, F, E, D, C, B, A) \leftarrow (G, F, E, D \oplus T_1, C, B, A, T_1 \oplus T_2)$
return $(V_1 \oplus A, ..., V_8 \oplus H)$

When a message P is longer than 512 bits, it is padded and divided into 512-bit chunks $\{P_0, ..., P_{n-1}\}$. Each chunk is processed through the compression function and chained thanks to a Merkle Damgård construction (see Figure 1). Finally, the hash Z corresponds to the output of the last compression function of the chain.

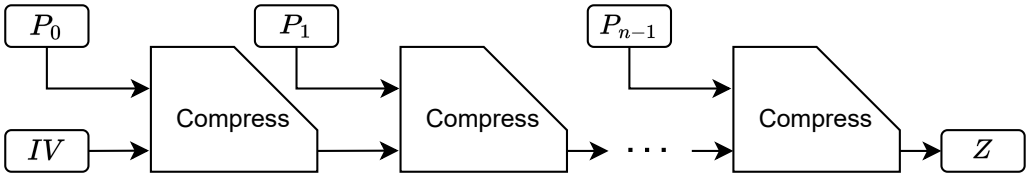


Figure 1: Merkle Damgård construction for SHA-2.

2.3 Keccak Specifications

SHA-3 is a hashing algorithm based on the Keccak-f permutation function [Dwo15]. 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 defined hereafter:

$$\theta : a[i][j][k] = a[i][j][k] \oplus \texttt{parity}(a[0..4][j-1][k]) \oplus \texttt{parity}(a[0..4][j+1][k-1])$$
$$\rho : a[i][j][k] = a[i][j][k-(t+1)(t+2)/2]$$
$$\pi : a[3i+2j][i][k] = a[i][j][k]$$
$$\chi : a[i][j][k] = a[i][j][k] \oplus \neg(a[i][j+1][k] \& (a[i][j+2][k])$$

with t indicating the current round, and ι defining the exclusive-or of the first word with a constant. These sub-routines 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 (see Figure 2). Keccak-f input of size b bits is divided into two parts: the "rate" r and the "capacity" $c = b - r$. The security level against collision and preimage attacks is half the size of c. 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 (see Figure 2). 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.

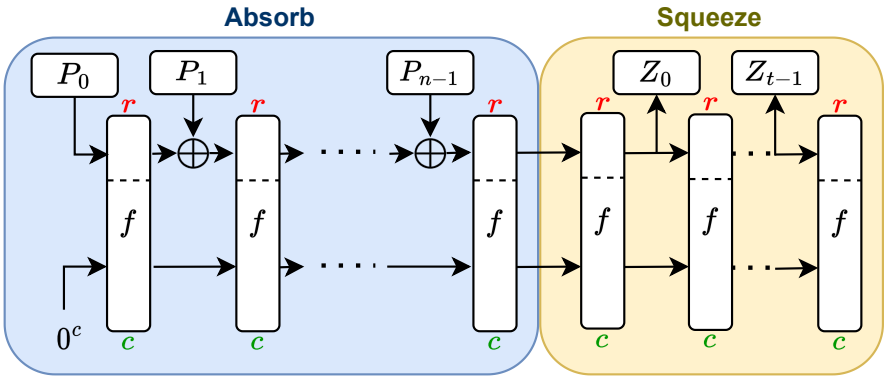


Figure 2: Keccak-f calls in Keccak.

Table 1: Algorithms that use Keccak and their corresponding parameters.

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

Several side-channel attacks targeting hashing functions have been presented in the literature. McEvoy *et al.* were the first to mount a *Differential Power Analysis* (DPA) based attack on a SHA-2 HMAC [MTMM07]. They consider a Hamming distance leakage model on a FPGA implementation. They describe masking countermeasures on SHA-2 internal routines that thwart their attack. In 2013, Belaïd *et al.* showed an improved attack under the Hamming weight leakage model necessitating less assumptions regarding the implementation [BBD+13]. They also exposed some countermeasures to cancel their attack paths. Note that both previous works do not recover the secret key, but rather the chaining value derived from the secret. The post-quantum cryptographic scheme SPHINCS was attacked with a DPA through its SHA-256 based pseudo random number generator by Kannwischer *et al.* [KGB+18]. More recently, Belenky *et al.* performed a template attack on a HMAC-SHA-2 hardware implementation [BDT+21].

Soft analytical side-channel attacks have been introduced by Veyrat *et al.* on an AES Furious implementation [VCGS14]. Interestingly, SASCA was shown to outperform *Algebraic Side-Channel Attacks* (ASCA), even in noise-free contexts [GS15]. Furthermore, 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 [PPM17, PP19, HHP+21, HSST23]. Kannwischer *et al.* mounted the first single trace SASCA on Keccak [KPP20]. 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 perform the θ transform of Keccak-f. Later, You and Kuhn developed a fully

bit-level approach of SASCA on Keccak [YK21]. Thanks to bit-level likelihoods gathered from a fragment template attack, they targeted a Keccak-f[1600] implementation on a Cortex-M4 device. Cassiers *et al.* implemented a *Regression based Linear Discriminant Analysis* (RLDA) [CDSU23] technique that allows templating 32-bit variables efficiently. They showed the soundness of their approach by attacking a 32-bit implementation of ASCON.

4 Attacker model

In this paper, we consider a profiled attack scenario. This means that we suppose that 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.*, in order to identify the leakage model or to select a set of points of interest for future templating) on a high number of intermediate variables of the target algorithm.

We consider the attacker to be able to craft templates for inter-round intermediate variables (including the hashing function input). Namely, our attacker model does not allow to template intra-round intermediate variables directly. This choice helps to tend towards a more fair comparison of the studied hash functions, while providing 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 [AA+14, SC17, MIV15], and for the compression rounds of SHA-2 [CKSV06, CL20]. Note that, in practice, a template 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 attacking phase to the observation of a single side-channel trace. This model can be relaxed depending on the use-case. For example, attacks presented in Section 7 and Section 8 can be conducted during the boot phase of a device. This allows reproducing the measurements with the exact same input data. In these cases, the attacker could use these multiple measurements in order to obtain better prediction accuracies on intermediate variables.

4.1 Leakage Simulation

As precised in Section 2, SHA-256 operates with 32-bit variables and Keccak[1600] handles 64-bit variables. Although the templating of 32-bit variables has been shown possible [CDSU23], manipulating probability distributions upon 2^{32} possible values requires a high amount of storage. This becomes completely impractical for 64-bit variables.

Another strategy is the so-called fragment template approach. This 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 [YK21] 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. Consequently, in this paper, we consider a bit-level leakage model to conduct our experiments. This model is particularly interesting when considering SHA-256 and Keccak-f, that essentially use bitwise operations in their internal routines. Namely, for each bit b, the leakage L is described as follows:

$$L = \alpha \cdot b + \beta,\ \beta \sim \mathcal{N}(\mu, \sigma^2) \tag{4}$$

where α is a multiplicative coefficient and β is a Gaussian noise parameter. Different levels of noise can then be simulated by modifying σ, the standard derivation of the noise distribution.

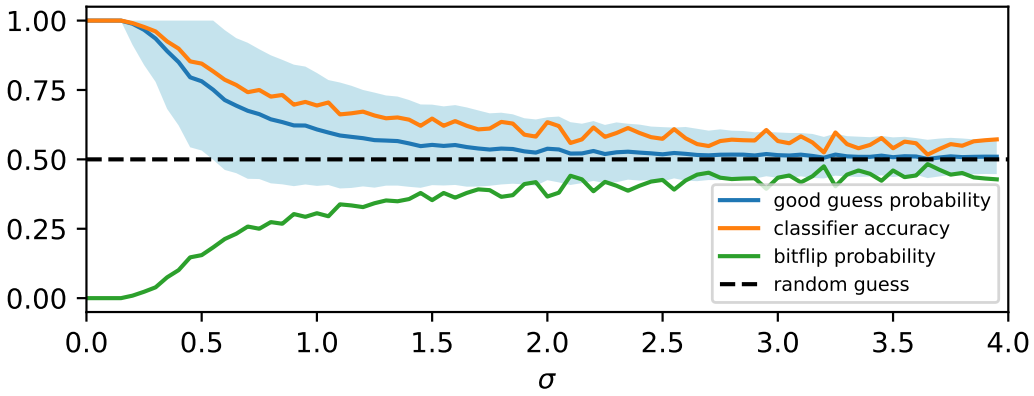


Figure 3: Bit leakage simulation statistics for increasing noise parameter σ, results are obtained with 1000 experiences per noise level. The orange curve depicts the average accuracy of a LDA classifier. The green curve represents the average bitflip probability. The blue curve illustrates the average probability of the good hypothesis, with standard deviation illustrated by the light-blue surface.

Fisher's *Linear Discriminant Analysis* (LDA) is selected as our classifier. This choice is sound in our simulated context because *(i)* the leakage is univariate, *(ii)* the simulated leakage model is linear and *(iii)* there are two classes to discriminate, with same covariance matrix by construction.

We perform a random sampling of leakages following Equation 4 with an increasing noise level. Statistics of the LDA classifier are depicted in Figure 3. As expected, when increasing the noise parameter σ, the accuracy of a LDA classifier converges towards a random guess (*i.e.*, 0.5).

Evaluation. We evaluate the accuracy of the attacks in this paper thanks to a "recovery ratio" that indicates the proportion of correctly recovered bits from the secret part of the hash function input, denoted S. Namely, considering $\#S$ as the number of secret bits, we have:

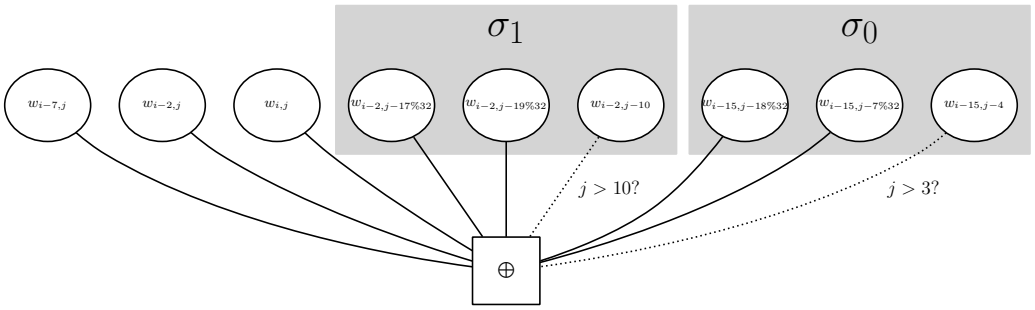
$$\text{recovery ratio} = \frac{\text{number of correct bits}}{\#S} \tag{5}$$

We choose to evaluate the recovery ratio instead of the success rate because the latter does not indicate the remaining quantity of the secret to be enumerated after the attack. In other terms, we believe that the recovery ratio is more suitable to indicate further practicality of key-enumeration algorithms [VCGRS13, Gro19, PSG16].

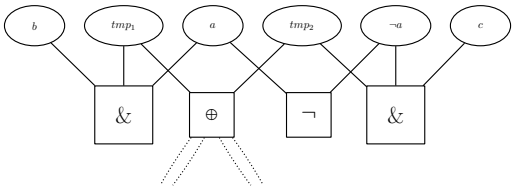
Baseline attack. To provide a comparison basis in the remainder of this paper, we introduce the notion of "baseline attack" which describes an attack that is only headed thanks to templating the input variables (without BP). For example, the accuracy of a baseline attack on a hashing function is obtained by only templating its input. The baseline allows to evaluate the correction capacity brought by SASCA.

Limitations. In this work, bit-level leakage modeling allows to lead attack simulations with varying noise level in order to evaluate the robustness of targeted algorithms against SASCA. Nevertheless, this bit-level model is a simplified version of the leakage models that can be observed in a real-case scenario.

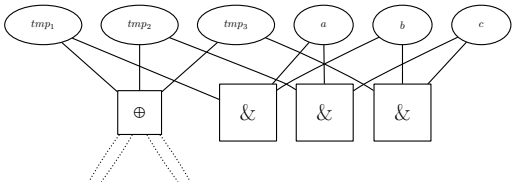
Furthermore, our SASCA approach relies on an underlying classifier that provides likelihoods to the probabilistic model. Hence, all advances and techniques in terms of



(a) subgraph for message expansion.



(b) subgraph for the Ch transform.



(c) subgraph for the Maj transform.

Figure 4: Bit-level subgraphs for SHA-256 message expansion, Ch and Maj routines

supervised side-channel analysis can be used to increase classification accuracy before being plugged into the probabilistic model.

5 Factor Graph Construction

5.1 SHA-256 Factor Graph

We build a factor graph where each bit of the SHA-256 intermediate states' 32-bit variables is considered as a variable node in a factor graph.

Variables. Hereafter, we describe the naming conventions for the variable nodes that are represented in the factor graph:

- The expanded message is composed of 64×32 variables denoted $w_{i,j}$, with $i \in \{0, ..., 63\}$ indicating the corresponding 32-bit word and $j \in \{0, ..., 31\}$ indicating each bit of such word in big-endian encoding (*e.g.*, $w_{1,0}$ is the most significant bit of the second 32-bit word of the expanded message).
- At each round t of the compression function, we define the intermediate state V^t as $(A^t, B^t, C^t, D^t, E^t, F^t, G^t, H^t)$. Each element of this tuple represents a set of 32 single bit variables.
- Variables corresponding to V^t are denoted $v^t_{i,j}$, with $i \in \{0, ..., 7\}$ (*e.g.*, $v^6_{3,4}$ denotes the fifth most significant bit of D^6.
- The input vector IV and the constants vectors K are defined as constants in the graph. Unlike variables, constants never update their marginal distributions.

Inner routines. The factor graph designed for SHA-256 can be viewed as a concatenation of several subgraphs. These subgraphs represent the inner logics of the message expansion and the compression rounds. First, the computation of a bit $w_{i,j}$, with $i < 16$, by the

message expansion function is represented with the subgraph illustrated in Figure 4a. This subgraph implements the σ_1 and σ_2 functions depicted in Subsection 2.2. We stress that the subgraphs of Σ_0 and Σ_1 are built analogously to the σ_0 and σ_1 parts in Figure 4a. Both Ch and Maj operations on 32-bit words are divided into 32 bit-level transforms (see Figure 4b and Figure 4c). Finally, the factor graph construction is parametrized in order to build a graph incorporating variables for the desired number of rounds.

Adding leakage. Adding leakage to the factor graph is performed by connecting an "observational factor" to the inter-round and expanded message variables. This factors' only function is to transmit the probability distribution corresponding to the template output.

5.2 Keccak-f Factor Graph

In this paper, we build a bit-level factor graph based on the work of You and Kuhn [YK21]. Namely, for each round of Keccak-f, the subgraph representing the θ routine incorporates nodes representing the parity intermediate variables C and D. Note that π and ρ routines are implemented with a simple wiring.

Adding leakage. Regarding the leakage, observational factors are linked to each round inputs and outputs. This differs from the approach taken in [YK21], where C and D related variables could also be connected to observational nodes, allowing explicit intra-round leakage exploitation. Finally, we allow the attacker to craft templates on the input variables of Keccak-f.

Standard instances. Recall that different Keccak[1600] standard instances exist. These different instances differ by the rate r and capacity c sizes used (see Figure 2). The versions studied in this paper are depicted in Table 1. In this work, when not precised otherwise, we consider by default the whole rate bits to constitute the secret. Formally, we consider $\#S = r$.

6 Single call attacks

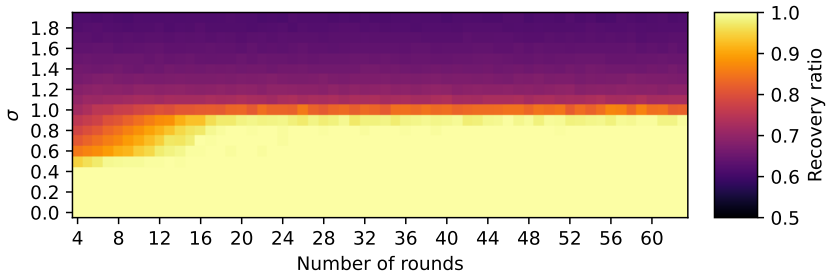
The aim of this section is to cover a general attack scenario where a side-channel attacker has the possibility to apply previously crafted templates upon a measurement of a single call to a hash function. Along this section, we aim at addressing the following questions: *(i)* What is the maximum noise level achievable for a successful attack? *(ii)* What is the minimum number of rounds to profile in order to obtain satisfactory recovery? *(iii)* Do structural aspects of SHA-256's compression function leads to non-homogeneous recovery of the input and how can it be compared to Keccak-f? And *(iv)* What accuracies can an attacker expect for different instances of Keccak and SHA-256?

6.1 Assessing the number of target rounds

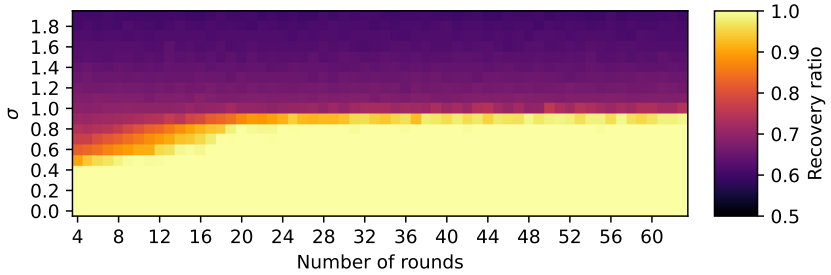
Firstly, the purpose of our analysis is to assess the minimal number of rounds that must be profiled by an attacker in order to get satisfactory recovery of the input considering noise. Indeed, finding such minimal number of rounds to be profiled allows reducing the number of variables to be templated by the attacker, with hopefully similar recovery potential. Hereafter, we evaluate this criterion through simulations at various noise levels for SHA-256 and Keccak.

6.1.1 SHA-256

For this simulation, the attacker is able to gain probabilistic information (*i.e.*, likelihoods originating from bit-level template attacks) on the whole expanded message W, as well as the output of several rounds' intermediate variables V. The chaining value is set to the default SHA-256 IV, which can either be known or unknown to the attacker. When unknown, IV can be templated by the attacker.



(a) SHA-256 compression function attack with a known IV.



(b) SHA-256 compression function attack with an unknown IV.

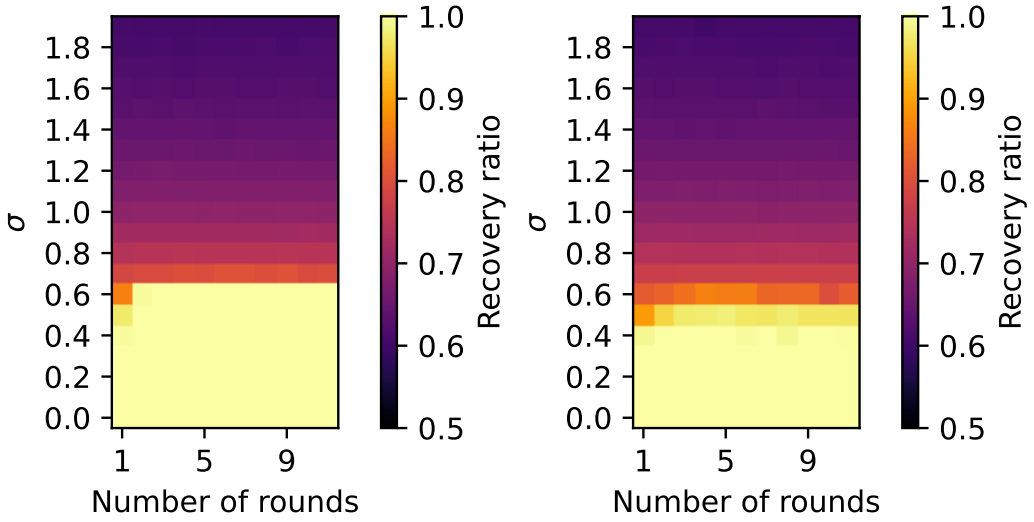
Figure 5: Recovery ratio for SHA-256 belief propagation attack considering variable rounds and noise levels with known (top) and unknown (bottom) chaining value IV.

The recovery ratio after SASCA on SHA-256 with increasing noise level and varying number of profiled rounds is depicted in Figure 5. We observe that *(i)* a recovery ratio of 1 is always achieved with a noise parameter inferior to 0.4, regardless the number of profiled rounds (superior to 3), *(ii)* increasing the number of profiled rounds above 20 does not allow to significantly increase the recovery capacity for high noise levels for both known IV scenario (see Figure 5a) and unknown IV scenario (see Figure 5b, and *(iii)* the knowledge of the IV is beneficial to SASCA, allowing perfect recovery at a higher noise level. Considering *(ii)*, further SASCA on SHA-256 will be performed with 20 profiling rounds.

6.1.2 Keccak-f

Similarly to the previous analysis, we aim at finding the number of Keccak-f rounds to be templated by an attacker. We perform two simulations with varying noise parameter σ and number of templated rounds. The first experiment evaluates the recovery ratio over the input for a SHA3-256/SHAKE-256 instance (see Figure 6a). The second experiment measures the recovery ratio for a fully unknown Keccak-f input state (see Figure 6b). This case corresponds to attacking a late invocation of Keccak-f in the sponge construction without prior knowledge of the output of the previous invocation.

In Figure 6 we can observe that attacking SHA3-256 allows to stay with a recovery ratio of 1 up to $\sigma = 0.6$, which is superior to what can be recovered for a full random input. Still, with $\sigma < 0.5$, an attacker can recover a high proportion of the full random input (superior



(a) Keccak-f[1600] with rate $r = 1088$ and capacity $c = 512$. This case corresponds to targeting the first Keccak-f call of a SHA3-256 instance.

(b) Keccak-f[1600] with full random input. This case corresponds to a late invocation of Keccak-f of any standard Keccak instance with $b = 1600$.

Figure 6: Recovery ratios after simulated BP attacks on Keccak-f[1600] with full unknown inputs (right) and $r = 1088$ unknown bits, the SHA3-256 FIPS standard (left).

to 0.9). We also notice that, regardless the targeted version, no significant improvement is brought by profiling more than 5 Keccak-f rounds. Further Keccak-f related analyses in this paper will be conducted considering templates on 5 rounds.

6.2 Error location

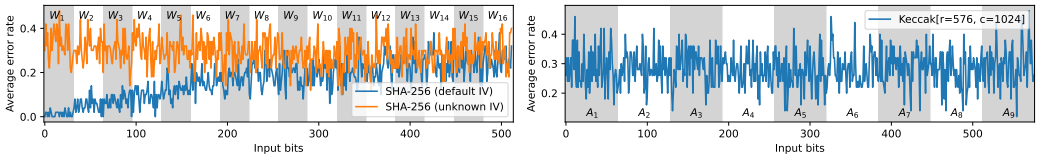
We investigate the location of recovery errors after the attack. In Figure 7a, we illustrate the error rate for each bit, averaged on 50 runs of the attack on SHA-256 for $\sigma = 1$. This error rate is measured for two instances: the "default IV" version where the attacker knows the chaining value IV, and the "unknown IV" version where the attacker does not know the chaining value, but has the possibility to craft bit-level templates on the latter.

Interestingly, in the "default IV" scenario, we observe that the error rate is increasing together with the index of 32-bit input words W_i. Recall from Algorithm 1 that the IV is manipulated together with K_1 (a public constant) and W_1 during the first compression round. Then, first round leakages of T_1, and hence D and A, are very informative regarding W_1. At the end of the first round, the 32-bit words vector (B, C, D, F, G, H) still contain known data. This reduces the entropy for T_1 and T_2 computations. The further the round, the more the uncertainty coming from the template attacks is spread in 32-bit words A to H. This explains the increasing error rate in the "default IV" case with respect to i, the index of the 32-bit input word W_i. Finally, after round 11, the uncertainty spreading makes the error rate of $W_i, i \geq 11$ similar to the "unknown IV" case.

Similar analysis is accomplished with Keccak-f with $r = 576$ unknown input bits and $c = 1024$ bits set to the default value 0. Results are depicted in Figure 7b. Unlike SHA-256, Keccak-f with partially known input shows no meaningful error location biases at a 64-bit word scale. This can be explained by the fact that Keccak-f is a permutation function, that efficiently blends known and unknown data, notably though θ, ρ, π and χ routines. Along with having good properties against cryptanalysis [MDA17], this structure has the advantage to quickly spread the uncertainty coming from the template attack.

Consequently, under the scenario of BP-based attacks, the compressive structure of

SHA-256 is a shortcoming regarding the confidentiality of the secret input. The "word by word" handling of the input with, at first, known data, gradually spreads the uncertainty coming from the templates, hence enabling a better recovery ratio on the first 32-bit words. As a result, small sized secrets processed as input of SHA-256 with known IV are more prone to be recovered when placed at the beginning of the input buffer. Interestingly, this case is encountered in SHA-256 based HMAC constructions. From a defensive point of view, if a compromise must be reached on a countermeasure against the attack proposed in this paper (countermeasures are later discussed in Section 10), protecting the first compression rounds must be a priority.



(a) Average error rate per SHA-256 input bit, computed for noise parameter $\sigma = 1.0$. Straps indicate 32-bit words of the input message.

(b) Average error rate per Keccak-f input bit, computed for noise parameter $\sigma = 1.0$. Straps indicate 32-bit words of the input message.

Figure 7: Average error rate per bit for SHA-256 and Keccak-f.

6.3 Evaluation of SHA-2 and SHA-3 attacks

We now compare the resistance of several SHA-2 and SHA-3 instances against SASCA. Each attack exploits templates applied upon 20 compression rounds for SHA-256 and 5 Keccak-f rounds for Keccak. For an increasing noise level, we run the attack 50 times on random inputs and we average the recovery ratio. 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 such simulations are illustrated in Figure 8.

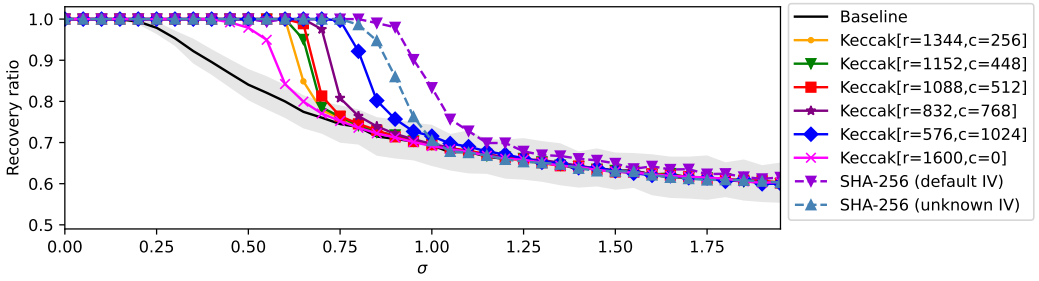


Figure 8: Comparison of the recovery ratios of a baseline classifier (without BP), SHA-2 and Keccak-f attacks. The gray surface represents the standard deviation of the baseline classifier's recovery ratio.

Benefits of SASCA. In Figure 8, we observe that SASCA provides better recovery ratios than baseline classifiers up to $\sigma = 0.75$ for all Keccak instances and up to $\sigma = 1.25$ for all SHA-256 instances. Beyond these noise levels, the SASCA approach does not seem to be significantly beneficial compared to the baseline attack. Finally, all attacks seem to converge to a recovery ratio of 0.5, which is equivalent to a random guess, when σ tends to infinity.

Known data proportion. Results provided in Figure 8 show that the proportion of known data plays an important role regarding the resistance of the attack to noise. Indeed, for Keccak instances, the success of the attack with higher noise levels is directly proportional to the size of the capacity c of the input, which is filled with zeros. Regarding SHA-256, we observe that the knowledge of the chaining value IV is beneficial to the attacker.

Comparing SHA-3 and SHA-256. In our setup, Keccak is globally more resistant to SASCA than SHA-256. Moreover, from an attacker point of view, the hardest SHA-2 instance (*i.e.*, SHA-256 with unknown input) can be attacked up to a higher noise level than the easiest SHA-3 instance (*i.e.*, Keccak[r=576, c=1024]).

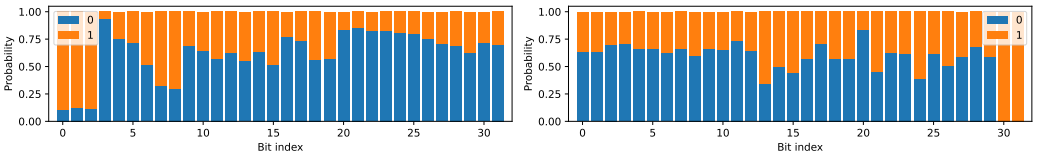
Comparison with You and Kuhn's work. We recall that You and Kuhn [YK21] managed to reach a success rate superior to 0.9 with SASCA on Keccak-f for all standard instances presented in Table 1, this on a software implementation running on a Cortex-M4 controller. Moreover, they reported a bit-level template attack accuracy of approximately 0.73. When reporting this value onto the baseline classifier recovery ratio curve in Figure 8, one can notice that only SHA3-512 instance (*i.e.*, Keccak[$r = 576$, $c = 1024$]) presents similar recovery ratios in our setup. This can be explained by several elements. Firstly, as precised in Section 4, we do not allow the attacker to build templates on intra-round intermediate variables (*e.g.*, C and D for Keccak-f, or $T1$ and $T2$ for SHA-256 compression function). Depending on actual implementation, templating these intra-round variables and including the obtained likelihoods into the graph in real-case attack scenarios could be beneficial to all attacks illustrated in Figure 8. Secondly, the results presented in [YK21] are derived from real-case attacks, with potential high-variability regarding template accuracies upon intermediate variables, due to the potential presence of different leakage models. Consequently, the set of template outputs for all intermediate variables are not compulsorily drawn from the same probability distribution. This differs with the work presented in this paper, where the same leakage model is considered for each variable.

6.4 Discussion

In this section, we presented a methodology to assess the minimal number of hashing function rounds necessary to mount successful SASCA on SHA-256 and SHA-3. Then, we identified biases regarding the error rate of SASCA induced by the compressive structure of SHA-2's internals. Noticeably, we showed that the first 32-bit words of the input can be recovered more easily with our attack. After, we analysed the benefits of SASCA compared to a baseline attack, and showed that the more input (and IV, in the particular case of SHA-256) is known, the easier it is for an attacker to recover the secret portion of the input with higher noise. For instance, this implies that Keccak instances with smaller rate r, while having higher security against cryptanalysis, are more sensitive to SASCA. Moreover, we showed that, with our setup, SHA-256 can be accurately attacked with SASCA up to higher noise levels than any considered SHA-3 instance. Furthermore, when comparing our simulated results with attacks previously proposed in the literature, we can assess that SHA-2 can be targeted with single trace SASCA in real-case scenarios.

7 Hashing on structured data

Until now, attack scenarios developed in this paper assumed that the target hashing function would take random data as input. However, when considering a firmware integrity check, the data processed by the hashing function consists in portions of a binary file partially composed of assembly instructions. The encoding of instructions is ruled by the relative ISA. Hence, the latter, as well as the implementation choices made by the compiler,



(a) ARMv7-A 32-bit instruction bits biases, bit index 0 indicates the least significant bit.

(b) RISC-V 32-bit instruction bits biases, bit index 0 indicates the least significant bit.

Figure 9: ARMv7-A and RISC-V instructions bit-level biases.

induce a structure on the binary file's data. In this section, we investigate the possible benefits of exploiting this structure from an attacker's perspective.

Bit-level biases. For this study, we aim at leveraging bit-level statistics from two sets of binary programs. We gather 1419 and 343 ARMv7-A and RISC-V binary files respectively. We disassemble all these binaries and only keep the 32-bit instructions contained in the `.text` section of the binary programs. This leads to datasets of approximately 27 and 7 million instructions for ARMv7-A and RISC-V, respectively.

For both datasets, each 32-bit instruction can be represented as $X = (x_i)_{0 \leq i < 32}$. Then, we want to compute the likelihood for each x_i such as $0 \leq i < 32$.

Statistics on per-bit instructions biases are depicted in Figure 9a and Figure 9b for ARMv7-A and RISC-V respectively. Firstly, we observe that both instruction sets lead to bitwise probability distributions that are not uniform. Moreover, we clearly observe the two most significant bits set to 1 systematically for the RISC-V instruction set, which is precised in the ISA specification.

Each bit prior distribution can be injected within the factor graph in order to take the bit-level biases into account while performing the attack.

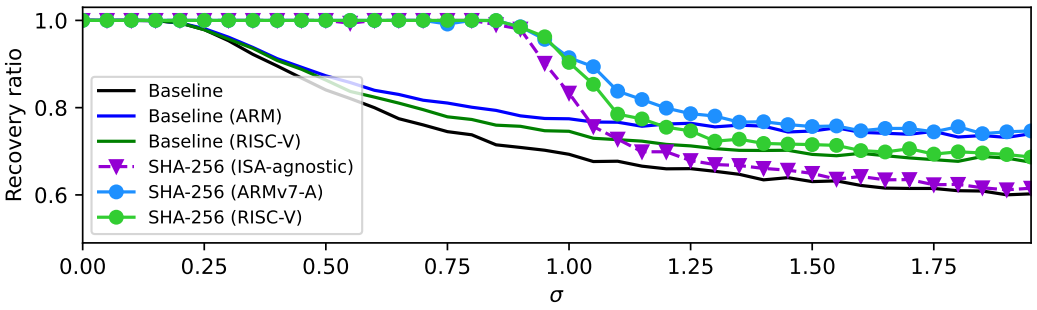


Figure 10: SHA-256 SASCA attacks considering ARM and RISC-V code input compared to baseline (*i.e.*, non SASCA) attacks and ISA-agnostic attack. For each noise level, the recovery ratio is averaged over 50 runs. The IV is known to the attacker.

Attack scenario. In order to perform the simulation, we randomly select a set of consecutive instructions within the code database and run the attack for increasing noise levels. Note that, for simplification purposes, we suppose the input of SHA-256 to be aligned to 32-bit words. Moreover, we consider the chaining value to be known: this corresponds to the "default IV" case scenario. In a real-case attack scenario, this would imply that targeting a compression phase in the Merkle Damgård scheme with prior knowledge of the preceding chaining value. The latter is either the known IV for the first compression call, or obtained by attacking the previous calls. In order to recover several compression input

blocks, the adversary can either chain the attacks, or, at the cost of fewer noise tolerance, perform attacks in an "unknown IV" scenario, as depicted in Subsection 6.3.

Simulation results. Results of this experiment are depicted in Figure 10. We observe that *(i)* considering either ARM or RISC-V priors allows a better recovery ratio compared to the ISA-agnostic baseline attack (default IV case in Figure 10), *(ii)* ARMv7-A biases provide better attack results than RISC-V for σ superior to 1, both for the baseline and SASCA approaches, and *(iii)* each attack accuracy converges towards the corresponding baseline attack accuracy when the noise increases. Note that if we target a 0.9 recovery ratio, ARMv7-A and RISC-V priors lead to the same noise resistance against SASCA (*i.e.*, $\sigma = 1$). Only higher noise levels highlight the superior contribution of ARMv7-A biases against RISC-V biases from an attacker's perspective.

Discussion. This experiment shows that when the secret input of a hashing function is structured, the latter can be exploited by an attacker in order to enhance the attack accuracy. When considering the specific case of a secure boot executed at each device startup, an attacker could reach higher template accuracies by exploiting side-channel measurements originating from multiple startups of the device. Finally, note that instructions coming from a firmware are a specific type of structured data. Indeed, the approach developed in this section could also be applied on other structured data such as JSON files, PDFs or images.

8 Targeting SHA-256 HMAC

In this section, we study the security implications of several calls to a hashing function with inputs derived from the same secret through the use-case of a SHA-256 HMAC. Precisely, we investigate how combining multiple invocation of a SHA-256 compression function can be beneficial from an attacker's perspective.

Attack principle. A HMAC provides authentication and integrity verification of the data P by using a key K and a hash function h. Formally, HMAC based on SHA-256 is computed as follows (see Figure 11):

$$\text{HMAC}_K = h\Big((K \oplus opad)||h\big((K \oplus ipad)||P\big)\Big) \tag{6}$$

One can observe that two derivations of K are processed in the SHA-256 compression function: $K \oplus ipad$ and $K \oplus opad$. Note that *ipad* and *opad* are known parameters of the scheme. In this case study, we consider a 128-bit key K. An attacker can perform a SASCA upon the two SHA-256 calls $h(K \oplus ipad)$ and $h(K \oplus opad)$ simultaneously by joining the two corresponding factor graphs with variables that represent the bits of K.

Simulation results. Simulation results are depicted in Figure 12a. We observe that combining the two graphs is beneficial to the attacker, as the recovery ratio is higher for the double graph attack than the simple graph one. As previous attacks developed in this paper, the advantage of the BP approach compared to a baseline classifier becomes less significant as the noise increases.

Noise and BP Convergence. In Figure 12b, we illustrate the recovery ratio and the number of loopy-BP iterations obtained for several runs of the double graph HMAC attack for an increasing noise level. Recall from Subsection 2.1 that a threshold on the maximal statistical change of marginals from one iteration to the next is setup as a stopping criterion. One can see that, for low noise levels (*i.e.*, inferior to $\sigma = 1$), the number of iterations for

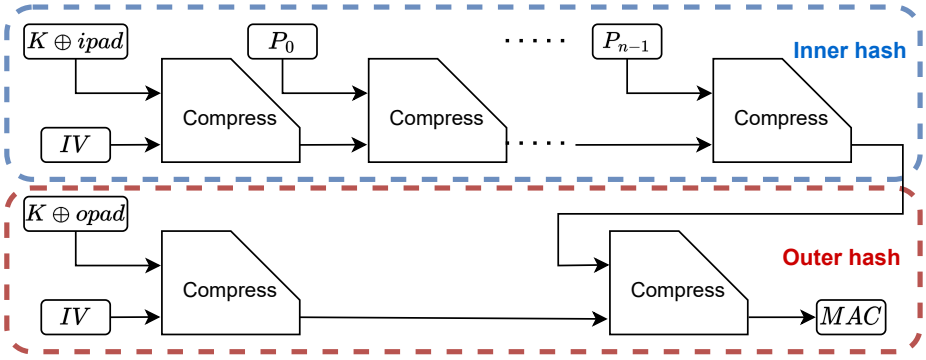
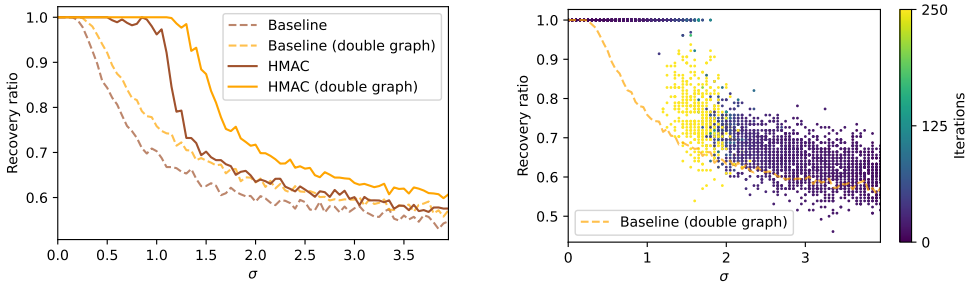


Figure 11: HMAC construction for SHA-256.



(a) Comparison of HMAC attacks, with simple and double graph variants, to baseline classifier.

(b) Number of iterations of the loopy-BP algorithm on the double graph attack.

Figure 12: HMAC attack accuracy with single and double graph and relationship between the noise level and the number of iterations.

all runs stays low (*i.e.*, inferior to 70) while the recovery ratio is at 1. This means that the loopy-BP algorithm converges quickly to the good hypothesis. From $\sigma = 1$ to $\sigma = 2$, the intermediate zone, the majority of runs that do not lead to a perfect recovery reach the maximum number of iterations (250 in this experiment), indicating that loopy-BP did not converge. Finally, after $\sigma = 2$, the number of iterations for all the runs drops below 70.

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 12b. The attacker, then, has the options of *(i)* trying to reach higher initial template accuracies, *(ii)* exploit the resulting marginal distributions to conduct a key enumeration strategy, or *(iii)* retry the attack later and hope to reach a low number of iterations of loopy-BP and reach a perfect recovery.

Discussion. The attack presented in this section shows that processing derivations of a secret value multiple times through a hashing function is beneficial from an attacker's perspective. Hence, such structures, that can be found in SHA-256 based HMAC, present a vulnerability allowing a side-channel attacker to extract secret data in presence of greater noise. Note that SHA-256 based HMAC is still widely used by default in several protocols and applications such as IPsec, SNMPv3, EIGRP or SSH.

One can remark that the presented HMAC attack does not rely on the data P. Consequently, if the same key K is used to compute the HMAC for several files, an attacker can extract the appropriate compression function calls for each file (*i.e.*, the ones taking $K \oplus ipad$ and $k \oplus opad$ as input) in order to enhance template accuracy (*e.g.*, by averaging the traces), or to link them in a wider graph.

Interestingly, if the attacker has the possibility to observe the processing of known data P through the HMAC function, they could perform a non-profiled attack as presented in [BBD+13] in order to recover the chaining value V_{out} outputted by $h(IV, K \oplus ipad)$. Then, as V_{out} is known and P_1 controlled by the attacker, the latter can predict intermediate values of $h(V_{out}, P_1)$ and consequently train models in order to obtain templates. Eventually, these templates can be applied in order to recover K with the attack presented in this section. Note that this scenario alleviates the attacker model, as the profiling and inference phases of the template attack can be performed on the same target device.

9 Targeting Kyber's encrypt function

Kyber is a post-quantum *Key Encapsulation Mechanism* (KEM) that has been standardized by the NIST in 2022 [ABD+19]. Kyber is a lattice-based scheme that aims at encapsulating a shared key for secure key exchange. Kyber's encapsulation 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.

Interestingly, to reach IND-CCA2 security, the receiver needs to perform a Fujisaki-Okamoto transform [FO99], 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 2. This algorithm takes as input a secret message (*i.e.*, the shared key to be recovered by the attacker). As one can see, a 32-bytes secret random coin r is derived thanks to a *Pseudo Random Function* (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 (see Appendix A for a proof of this statement).

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 2, r is manipulated with $PRF(r, N)$, for $N \in \{0, ..., 2k\}$, with $PRF(k, N) = \text{SHAKE-256}(r||N)$. 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 (see Table 3 in Appendix A). Note that the higher N_{tot}, the higher is Kyber's security level.

We create a joint factor graph that represents intermediate variables of all N_{tot} SHAKE-256 calls. The factor graph has bit-level variables that represent the 256-bit secret coin r. This structure allows to aggregate the results originating from a template attack performed on each SHAKE-256 call.

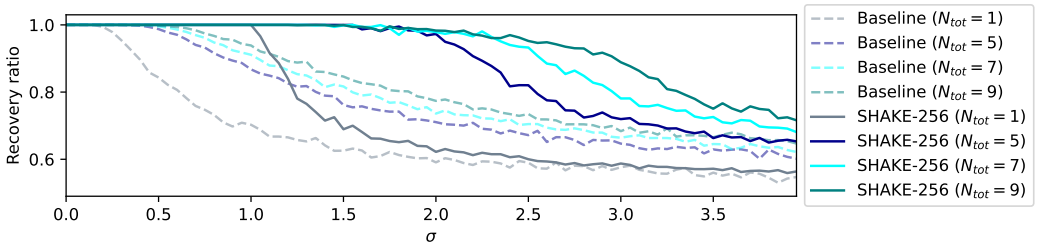


Figure 13: Average recovery ratio over 50 runs with increasing noise on Kyber's encryption attack considering multiple SHAKE-256 calls.

Algorithm 2: Kyber encryption function [ABD+19].

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 $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$

$N \leftarrow 0$
$\hat{\mathbf{t}} \leftarrow \texttt{decode}_{12}(pk)$
$\hat{\mathbf{A}} \leftarrow \texttt{generate\_public\_matrix}(pk)$
for i *from* 0 *to* $k-1$ **do**
  $\mathbf{r}[i] \leftarrow CBD_{\eta_1}(PRF(r, N))$
  $N \leftarrow N + 1$
for i *from* 0 *to* $k-1$ **do**
  $\mathbf{e_1}[i] \leftarrow CBD_{\eta_2}(PRF(r, N))$
  $N \leftarrow N + 1$
$e_2 \leftarrow CBD_{\eta_2}(PRF(r, N))$
$\hat{\mathbf{r}} = NTT(\mathbf{r})$
$\mathbf{u} = NTT^{-1}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \mathbf{e_1}$
$v = NTT^{-1}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + e_2 + \texttt{decompress}_q(\texttt{decode}_1(m), 1)$
$c_1 = \texttt{encode}_{d_u}(\texttt{compress}_q(\mathbf{u}, d_u))$
$c_2 = \texttt{encode}_{d_v}(\texttt{compress}_q(v, d_v))$
return $c = (c_1 || c_2)$

Simulation results. Attack results with increasing level of noise are illustrated in Figure 13. 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 8. 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, denoted 10^*1, to fill the rate part of the input of Keccak-f. This leads to 1344 known bits and $\#S = 256$ unknown bits within the latter input. The size of the manipulated secret being smaller than all previously analysed Keccak instances, the higher accuracy observed for the SHAKE-256 ($N_{tot} = 1$) instance follows the conclusions brought in Section 6.

Most importantly, analogously to the results presented in Section 8, considering several calls to SHAKE-256 sharing the same secret allows to drastically increase the level of noise supported by the attack, folliwing the evolution of N_{tot}.

10 Countermeasures

In this section we present several countermeasures suggestions to cope with the attacks presented in this paper.

Protocol. In Section 8 and Section 9, 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 could be headed towards implementing derivations from a secret that are harder to exploit from an attacker's perspective. Finally, as seen in Subsection 6.2, placing SHA-256 HMAC's secret key at the end of the input buffer makes harder its recovery with SASCA.

Masking. All attacks presented in this paper rely on the ability for an attacker to craft a template attack on several intermediate variables within the targeted hashing functions. Appropriate masked implementations of Keccak have already been presented in the literature [GSM17, ABP+18]. Moreover, hardware implementations of the SHA-2 HMAC scheme with a masked compression function have been proposed [HWZ18]. We showed that the first rounds of SHA-256 compression function must be protected in priority because secret data is mixed with known data. An adaptive masking (*e.g.*, masking only the n first compression rounds) could be considered to thwart our proposed attack with limited impact.

Shuffling. Randomizing the processing order of compression intermediate variables in SHA-256, or shuffling Keccak-f inputs represents a promising countermeasure. Nevertheless, there exist data dependencies (*e.g.*, for SHA-256 message expansion) that limit the capacity to shuffle the operations.

Hiding. All attacks presented in this paper are valid until a certain level of noise. Artificial extra noise and jitter can be foreseen as a countermeasure for the attacks we propose here. Hence, classical measures that tend to limit the power consumption variations or that insert dummy cycles during the sensitive code execution can be beneficial. Indeed, these measures would complicate the template construction phase of the attack, and thus lower the SASCA accuracy.

11 Conclusion and Further Work

Table 2: Supported noise levels where the average recovery ratio is over 0.9

Scenario	$\#S$	σ
SHA-256 (Default IV)	512	0.95
SHA-256 (Unknown IV)	768	0.85
Keccak[r=1600, c=0]	1600	0.55
Keccak[r=1344, c=256]	1344	0.60
Keccak[r=1152, c=448]	1152	0.65
Keccak[r=1088, c=512]	1088	0.65
Keccak[r=832, c=768]	832	0.70
Keccak[r=576, c=1024]	576	0.80
SHA-256 (ARM priors)	512	1.0
SHA-256 (RISCV priors)	512	1.0
SHA-256 (HMAC)	128	1.1
SHA-256 (HMAC double graph)	128	1.4
Kyber's encrypt SHAKE-256 ($N_{tot} = 1$)	256	1.1
Kyber's encrypt SHAKE-256 ($N_{tot} = 5$)	256	2.2
Kyber's encrypt SHAKE-256 ($N_{tot} = 7$)	256	2.55
Kyber's encrypt SHAKE-256 ($N_{tot} = 9$)	256	2.95

In this paper, we investigated the security of SHA-256 and SHA-3 hashing functions against soft analytical side-channel attacks that aim at retrieving a secret input under a bit-level leakage model. In particular, we showed that SHA-3 is more resistant to SASCA than SHA-256. This is partially due to the compressive structure of SHA-256 that generates a bias during the first compression rounds by mixing known and unknown data. Such bias allows SASCA to recover more efficiently the first 32-bit words of the input. Then, through a boot image integrity check use-case, we experimentally showed

that the underlying structure of hashed inputs (assembly instructions in this example) can be exploited in order to enhance SASCA accuracy. Next, we mounted new attacks that exploit multiple calls to a hashing function that process data derived from the same secret. This approach was evaluated on SHA-256 based HMAC and Kyber's encryption function, that is based on SHAKE-256. This type of construction leads to additional vulnerabilities when considering an attacker with SASCA potential. In Table 2, we summarize the maximum noise levels that lead to 90% input recovery ratio through SASCA for all attacks presented in this paper. Particularly, it shows that attacks based on multiple hash function invocations really push forward the acceptable noise level from an attacker's perspective. In cases where side-channel attacks represent a threat, such vulnerability must be taken into account when designing new schemes that use hashing function to manipulate secret data, particularly by examining appropriate countermeasures. For instance, post-quantum schemes such as SPHINCS, that are mostly based on hash functions, should be scrutinized in this regard.

In this paper, we deliberately eluded the practical difficulties of performing a template attack on hashing functions. 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.

References

[AA+14] Alia Arshad, Arshad Aziz, et al. Compact implementation of SHA3-512 on FPGA. In *2014 Conference on Information Assurance and Cyber Security (CIACS)*, pages 29–33. IEEE, 2014.

[ABD+19] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-kyber algorithm specifications and supporting documentation. *NIST PQC Round*, 2(4):1–43, 2019.

[ABP+18] Victor Arribas, Begül Bilgin, George Petrides, Svetla Nikova, and Vincent Rijmen. Rhythmic Keccak: SCA security and low latency in HW. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 269–290, 2018.

[BBD+13] Sonia Belaïd, Luk Bettale, Emmanuelle Dottax, Laurie Genelle, and Franck Rondepierre. Differential power analysis of HMAC SHA-2 in the Hamming weight model. In *2013 International Conference on Security and Cryptography (SECRYPT)*, pages 1–12, July 2013.

[BDT+21] Yaacov Belenky, Ira Dushar, Valery Teper, Hennadii Chernyshchyk, Leonid Azriel, and Yury Kreimer. First full-fledged side channel attack on HMAC-SHA-2. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 31–52. Springer, 2021.

[CDSU23] Gaëtan Cassiers, Henri Devillez, François-Xavier Standaert, and Balazs Udvarhelyi. 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*, pages 270–293, 2023.

[CKSV06] Ricardo Chaves, Georgi Kuzmanov, Leonel Sousa, and Stamatis Vassiliadis. Improving SHA-2 hardware implementations. In *Cryptographic Hardware*

and Embedded Systems-CHES 2006: 8th International Workshop, Yokohama, Japan, October 10-13, 2006. Proceedings 8, pages 298–310. Springer, 2006.

[CL20] Yimeng Chen and Shuguo Li. A high-throughput hardware implementation of SHA-256 algorithm. In *2020 IEEE international symposium on circuits and systems (ISCAS)*, pages 1–4. IEEE, 2020.

[Dwo15] Morris J Dworkin. SHA-3 standard: Permutation-based hash and extendable-output functions. 2015.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Annual international cryptology conference*, pages 537–554. Springer, 1999.

[Gro19] Vincent Grosso. Scalable key rank estimation (and key enumeration) algorithm for large keys. In *Smart Card Research and Advanced Applications: 17th International Conference, CARDIS 2018, Montpellier, France, November 12–14, 2018, Revised Selected Papers 17*, pages 80–94. Springer, 2019.

[GS15] Vincent Grosso and François-Xavier Standaert. ASCA, SASCA and DPA with enumeration: which one beats the other and when? In *Advances in Cryptology–ASIACRYPT 2015: 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29–December 3, 2015, Proceedings, Part II 21*, pages 291–312. Springer, 2015.

[GSM17] Hannes Groß, David Schaffenrath, and Stefan Mangard. Higher-order side-channel protected implementations of keccak. In *2017 Euromicro Conference on Digital System Design (DSD)*, pages 205–212. IEEE, 2017.

[HHP+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 Transactions on Cryptographic Hardware and Embedded Systems*, pages 88–113, 2021.

[HSST23] Julius Hermelink, Silvan Streit, Emanuele Strieder, and Katharina Thieme. Adapting belief propagation to counter shuffling of NTTs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 60–88, 2023.

[HWZ18] Zhenhao He, Liji Wu, and Xiangmin Zhang. High-speed pipeline design for hmac of sha-256 with masking scheme. In *2018 12th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID)*, pages 174–178. IEEE, 2018.

[KFL01] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519, 2001.

[KGB+18] Matthias J Kannwischer, Aymeric Genêt, Denis Butin, Juliane Krämer, and Johannes Buchmann. Differential power analysis of XMSS and SPHINCS. In *Constructive Side-Channel Analysis and Secure Design: 9th International Workshop, COSADE 2018, Singapore, April 23–24, 2018, Proceedings 9*, pages 168–188. Springer, 2018.

[KPP20] Matthias J Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on keccak. *Cryptology ePrint Archive*, 2020.

[MDA17] Silvia Mella, Joan Daemen, and Gilles Van Assche. New techniques for trail bounds and application to differential trails in Keccak. *IACR Transactions on Symmetric Cryptology*, pages 329–357, March 2017.

[MIV15] Harris E Michail, Lenos Ioannou, and Artemios G Voyiatzis. Pipelined SHA-3 implementations on FPGA: Architecture and performance analysis. In *Proceedings of the Second Workshop on Cryptography and Security in Computing Systems*, pages 13–18, 2015.

[MTMM07] Robert McEvoy, Michael Tunstall, Colin C Murphy, and William P Marnane. Differential power analysis of HMAC based on SHA-2, and countermeasures. In *Information Security Applications: 8th International Workshop, WISA 2007, Jeju Island, Korea, August 27-29, 2007, Revised Selected Papers 8*, pages 317–332. Springer, 2007.

[PP19] Peter Pessl and Robert Primas. More practical single-trace attacks on the number theoretic transform. In *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 6*, pages 130–149. Springer, 2019.

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. 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*, pages 513–533. Springer, 2017.

[PSG16] Romain Poussier, François-Xavier Standaert, and Vincent Grosso. Simple key enumeration (and rank estimation) using histograms: An integrated approach. In *Cryptographic Hardware and Embedded Systems–CHES 2016: 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings 18*, pages 61–81. Springer, 2016.

[SC17] Magnus Sundal and Ricardo Chaves. Efficient FPGA implementation of the SHA-3 hash function. In *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 86–91. IEEE, 2017.

[VCGRS13] Nicolas Veyrat-Charvillon, Benoît Gérard, Mathieu Renauld, and François-Xavier Standaert. An optimal key enumeration algorithm and its application to side-channel attacks. In *Selected Areas in Cryptography: 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers 19*, pages 390–406. Springer, 2013.

[VCGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In *Advances in Cryptology–ASIACRYPT 2014: 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, ROC, December 7-11, 2014. Proceedings, Part I 20*, pages 282–296. Springer, 2014.

[YK21] Shih-Chun You and Markus G Kuhn. Single-trace fragment template attack on a 32-bit implementation of keccak. In *International Conference on Smart Card Research and Advanced Applications*, pages 3–23. Springer, 2021.

A Kyber encrypt attack soundness

By looking backwards at Algorithm 2, 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:

$$v' = \texttt{decompress}_q(\texttt{decode}_{d_v}(c_2), d_v) \tag{7}$$

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

$$\delta_{err} = |v - v' mod^{\pm} q| \leq \left\lceil \frac{q}{2^{d_v+1}} \right\rfloor \tag{8}$$

with $\lceil x \rfloor$ being the rounding operation. The value of δ_{err} corresponding to each security level of Kyber is depicted in Table 3.

By setting $m_{dec} = \texttt{decompress}_q(\texttt{decode}_1(m), 1)$, from Algorithm 2 we have:

$$v = NTT^{-1}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + e_2 + m_{dec} \tag{9}$$

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:

$$m'_{dec} = v' - \left(NTT^{-1}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + e_2\right) \tag{10}$$

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

$$\texttt{compress}_q(m'_{dec}, 1) = \left\lceil \frac{2}{q} \cdot m_{dec} \right\rfloor \mathrm{mod}^{+} 2 \tag{11}$$

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:

$$m = \texttt{encode}_1(\texttt{compress}_q(m'_{dec}, 1)) \tag{12}$$

Table 3: 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