IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2017, No. 1, pp. 80–105. DOI:10.13154/tosc.v2017.i1.80-105

ISAP – Towards Side-Channel Secure Authenticated Encryption

Christoph Dobraunig, Maria Eichlseder, Stefan Mangard,
Florian Mendel and Thomas Unterluggauer

Graz University of Technology, Graz, Austria
firstname.lastname@iaik.tugraz.at

Abstract. Side-channel attacks and in particular differential power analysis (DPA) attacks pose a serious threat to cryptographic implementations. One approach to counteract such attacks are cryptographic schemes based on fresh re-keying. In settings of pre-shared secret keys, such schemes render DPA attacks infeasible by deriving session keys and by ensuring that the attacker cannot collect side-channel leakage on the session key during cryptographic operations with different inputs. While these schemes can be applied to secure standard communication settings, current re-keying approaches are unable to provide protection in settings where the same input needs to be processed multiple times.
In this work, we therefore adapt the re-keying approach and present a symmetric authenticated encryption scheme that is secure against DPA attacks and that does not have such a usage restriction. This means that our scheme fully complies with the requirements given in the CAESAR call and hence, can be used like other nonce-based authenticated encryption schemes without loss of side-channel protection. Its resistance against side-channel analysis is highly relevant for several applications in practice, like bulk storage settings in general and the protection of FPGA bitfiles and firmware images in particular.

Keywords: authenticated encryption · fresh re-keying · passive side-channel attacks · sponge construction · permutation-based construction

1 Introduction

Motivation. Passive side-channel attacks and in particular differential power analysis (DPA) pose a serious threat to the security of cryptographic implementations. These attacks allow to learn information about the secret key that is processed in a device by observing physical properties, like the power consumption [KJJ99] or the electromagnetic (EM) field [QS01]. They are a threat whenever a device performs cryptographic operations with a key that is not known to the holder of a device. This is the case, for example, when a sensor device is installed in a non-protected area to communicate data to some backend, when a manufacturer performs an encrypted firmware update on devices in the field, when a device working on encrypted data is lost, or when a device is rented by one party to another.

While passive side-channel attacks have mainly been a threat to ATM and pay TV cards at the time of their publication, these attacks are now relevant to a wide range of devices of the Internet of Things (IoT). A recent example is the IoT attack by Ronen et al. [ROSW16], where adjacent Philips Hue smart lamps infect each other with a worm that has the potential to control the device. One crucial part of this attack is the recovery of the global AES-CCM key that is used to encrypt and verify firmware updates with the help of a sophisticated DPA attack. As another prominent example, the keys for

Received: 2016-11-23, Accepted: 2017-01-23, Published: 2017-03-08

FPGA bitfile encryption of several generations of FPGAs have been revealed by DPA attacks [MBKP11, MS16].

DPA attacks are the most powerful passive side-channel attacks in practice. They accumulate information about a cryptographic key by observing multiple en-/decryptions of different inputs. The fact that different inputs are used allows statistical techniques, like Bayesian distinguishers [CRR02] or correlation techniques [BCO04], to extract keys very efficiently.

While these attacks typically require a standard oscilloscope, there are now also open source projects for attack setups on software implementations [OC14]. Unprotected software implementations of cryptographic algorithms typically can be broken by observing less than 100 en- or decryptions with a key [MOP07]. Given the low effort of the attacks, there is great need for countermeasures.

State of the Art. In order to protect cryptographic keys against side-channel attacks, a lot of research has been conducted during the last two decades. Today, there essentially exist two approaches to counteract the attacks. The first approach works by hardening the implementation of cryptographic algorithms with techniques like hiding [MOP07] or masking [PR13]. The drawback of this approach is that the overhead for securing a cryptographic primitive against side-channel attacks is very high and depends on the cryptographic primitive itself. Therefore, in the past several ciphers have been proposed to reduce this cost. For example, the authenticated ciphers Ascon [DEMS14], Ketje/Keyak [BDP+14a, BDP+14b], PRIMATES [ABB+14], and SCREAM [GLS+14] of the ongoing CAESAR competition [CAE14] have all been designed with this goal in mind. However, the protected implementation of these designs still leads to a significant overhead and the cost of masking still increases significantly with the protection order [ISW03].

The second approach to counteract side-channel attacks is to change cryptographic protocols in such a way that certain types of side-channel attacks cannot be performed at all on the underlying cryptographic primitive. In particular, if the protocol design inherently prevents DPA attacks, the underlying cryptographic primitive only needs to be secured against attacks that extract information about the key by observing cryptographic operations for a single fixed input. Following the definitions in [MOP07], we refer to the class of attacks that require to observe a device processing the same or a few inputs as simple power analysis (SPA), whereas we refer to the class of attacks that require to observe a device processing many different inputs under the same key as differential power analysis (DPA). A protected implementation of the primitive against SPA attacks induces a significantly lower overhead than against DPA attacks. An example of such an approach of inherently preventing DPA attacks is fresh re-keying [MSGR10, MPR+11, BDH+14, DKM+15] and leakage-resilient cryptography, which brought forth encryption schemes [Pie09, FPS12], message authentication codes (MACs) [PSV15] and authenticated encryption schemes [BKP+16].

Schemes with inherent protection against DPA attacks require a side-channel secure initialization in order to obtain a fresh session key for every cryptographic operation. This session key is typically derived from a pre-shared master key using a nonce. The purpose of the secure initialization is to ensure that cryptographic operations for different data inputs are always done using different keys. Hence, whenever a party encrypts or authenticates data, a new nonce has to be generated to derive a new session key.

While this effectively prevents DPA attacks on the sender's encryption or authentication process, the situation is more challenging for the receivers who perform decryptions or verifications. While the sender can generate the nonce and thus ensure that session keys are always fresh, the receiver must process any data he receives, with no control over the nonce. In order to prevent DPA attacks in these cases, one possible approach is that all communicating parties contribute to the nonce that is used to derive the session key from

a pre-shared master key [MPR+11]. This prevents an attacker from collecting side-channel information for the decryption of several different ciphertexts under the same nonce (and thus the same session key). However, this approach requires additional communication or synchronization between parties, which is often not possible in practice.

Our Contribution. We propose Isap, a symmetric authenticated encryption scheme that is designed to prevent DPA on both encryption and decryption. Isap fulfills all functional requirements for nonce-based authenticated encryption as defined by the CAESAR call [CAE14] and at the same time provides protection against DPA attacks for all involved parties. In addition, Isap limits the attack surface against decryption to SPA attacks and thus might be the first step towards a fully side-channel secure authenticated encryption scheme, addressing an open research problem mentioned by Pereira et al. [PSV15] and Berti et al. [BKP+16].

One of the main observations is that verifying authenticity before decryption protects the decryption procedure from DPA attacks, whereas the verification itself can be protected by a suitable derivation of the authentication session key. In addition, we show that sponges provide an elegant way to argue the resistance of permutation-based designs to SPA attacks. This flexibility motivates the fact that all building bocks of Isap are based on sponges.

The results of our hardware implementation show that the concrete instances Isap-128 and Isap-128a (that are based on 400-bit Keccak permutations) can be implemented in a straightforward manner with an area of 14 kGE, with the benefit compared to existing schemes that Isap provides DPA security up to the same order as the used re-keying function even for multiple decryption.

Open Questions. Isap protects against DPA and is designed to cope with limited SPA leakage. However, we still require dedicated countermeasures against SPA on implementation level. Such countermeasures are particularly crucial for the decryption unit, since the same data can be decrypted multiple times. This may reduce the measurement noise, making an SPA attack easier. Quantifying the SPA leakage of an implementation remains an open problem in practice. Another open question concerns the formal verification of our side-channel assumptions. While a security proof using state-of-the-art concepts of leakage-resilient cryptography might be out of reach, since Isap allows multiple decryption of the same data without introducing new randomness, it is still an open question if parts of our scheme or some specific properties like its resistance against DPA attacks can be formally proven.

Outline. We first recall the idea and limitations of fresh re-keying in Section 2. In Section 3, we specify the sponge-based authenticated cipher Isap. We give the design rationales of Isap in Section 4, and analyze its security in Section 5. Finally, we provide implementation results in Section 6 and conclude in Section 7.

2 Background to Re-keying

While cryptographic implementations can be protected via mechanisms like hiding or masking, frequent re-keying is a countermeasure to DPA that can be seen to work on protocol level. The idea of frequent re-keying is to prevent DPA on the cryptographic primitive by limiting the number of processed inputs per key. In other words, it limits the data complexity for each key by a small number q that renders DPA on the key infeasible (q-limiting [SPY+10]). It is nowadays a common assumption that small data complexities, i.e., $q = 1$ and $q = 2$, have sufficiently small side-channel leakage and do not allow for successful key recovery from DPA attacks [BDH+14, Pie09, SPY+10, TS15].

Frequent re-keying was first proposed for protecting embedded devices such as RFID tags [MSGR10, Koc03]. On the encryption of every new plaintext P, the block cipher E is provided with a new session key K^*. This session key K^* is derived from a pre-shared master secret K and a nonce N that is randomly generated on the tag. This inherently prevents DPA on the session key K^* of the block cipher E. However, for key derivation it requires a re-keying function $g : (K, N) \mapsto K^*$ that is easy to protect against both SPA and DPA attacks.

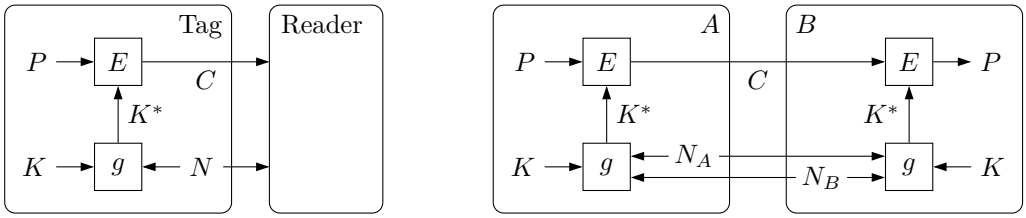
2.1 Secure Re-Keying Function

A secure re-keying function $g : (K, N) \mapsto K^*$ derives a new session key K^* from a master key K and a fresh nonce N and needs to be secure against both SPA and DPA attacks. This security against side-channel attacks can be achieved either on an algorithmic level, or by countermeasures for implementations. Hence, several options for choosing and implementing secure re-keying functions have been proposed.

For instance one option is to build g in such a way that it is easy to secure by classical countermeasures like masking. This is the basic idea of fresh re-keying described in [MSGR10, MPR+11], which uses a polynomial multiplication of K and N to implement g. This multiplication can be masked easily. However, as pointed out in [BFG14, BCF+15, GJ16, PM16], the algebraic structure of a multiplication opens the door to attacks on g and the encryption. Recently, this issue has been addressed by Dziembowski et al. [DFH+16], who propose two new schemes based on learning parity with leakage and learning with rounding.

A second option presented in [SPY+10, FPS12] is based on the classical GGM construction [GGM86]. The GGM construction can be used to mix a secret K with a public N in a tree-like approach, where on each tree level, exactly one bit of the public N is absorbed. Starting with $s_0 = K$, the key s_{i+1} is computed by encrypting one of two predefined plaintexts P_0, P_1 with the key s_i, depending on the i-th bit of N. The output of the last level is then, after postprocessing, used as the session key K^*. In this approach, an attacker only obtains the leakage for two inputs P_0 and P_1 to collect information about each s_i. The construction is thus 2-limiting and is usually considered to be secure against DPA.

Another option presented in [MSJ12, BDH+14] also originates from the classical GGM construction. It follows the idea of [SPY+10] by extending the number of observable measurements per key and deriving a leakage-resilient pseudo-random function (LR-PRF) from common block cipher designs to achieve secure re-keying. The main assumption of this approach is that the attacker is not able to distinguish the leakage of different hardware components on a chip.



(a) Protecting one party.

(b) Protecting both parties.

Figure 1: Re-keying of block ciphers.

2.2 Limitations and Open Problems

One major problem of re-keying schemes such as in Figure 1a is that the reader remains vulnerable to DPA attacks. For instance, such a re-keying scheme can successfully prevent DPA attacks on a device that solely performs encryption or authentication of messages, i.e., the sender of a message, but fails to protect a device performing decryptions or verifications, i.e., the receiver of a message. This is caused by the lack of control of a decryption device on the nonce N and allows attackers to send arbitrary messages to the decryption device using the same nonce N for all sent messages. This malicious procedure results in different messages being decrypted using the same session key K^*. As a result, decryption is vulnerable to DPA, and more concretely, it is the multiple decryption with the same session key K^* that causes this DPA vulnerability. This problem of securing decryption against side-channel attacks was also mentioned by Pereira et al. [PSV15] and Berti et al. [BKP+16].

In order to prevent this kind of DPA attacks, the receiver either needs to be protected by other means [MSGR10], or the receiver needs to be stateful in order to prevent decryption with the same session key twice, or all communication parties are required to contribute to the nonce that is used to derive the session key from a pre-shared master key [MPR+11] as shown in Figure 1b. However, the requirement of both sender and receiver being stateful bears some practical downsides ranging from synchronisation issues between sender and receiver to potential denial-of-service attacks, e.g., if the nonce is a counter and the receiver rejects all messages with a nonce smaller than the last valid nonce which is stored at the receiver. Also the option that all communication parties are required to contribute to the nonce is impractical in several prominent use cases, such as unidirectional/broadcast communication and encrypted storage. Recently, the need for DPA protection in these settings has been pointed out by attacks targeting the decryption of firmware images [ROSW16], or FPGA bitfiles [MBKP11, MS16]. While it is impossible to let a receiver contribute to the nonce in unidirectional communication settings, the additional overhead of letting each receiver contribute to the nonce in a broadcast setting could potentially make an application unpractical. In encrypted storage, the receiving device simply cannot contribute to the nonce, but must be able to decrypt the encrypted data, e.g., an encrypted FPGA bitfile, in all situations and possibly multiple times. To maintain DPA security in this case, one idea would be to re-encrypt the stored data whenever it is read. In practice, however, this is often not possible, e.g., due to the limited number of write operations in flash memory. Moreover, repeated re-encryption can eventually result in a loss of confidentiality [UWM17]. In the next section, we therefore present Isap, a new authenticated encryption scheme that is also secure against DPA attacks in these scenarios.

3 Specification of ISAP

Isap is a family of authenticated ciphers focusing to be secure against passive side-channel attacks. Its functional interface is the same as specified by the CAESAR competition for authenticated encryption [CAE14]: Isap encrypts a plaintext P to a ciphertext C. Additionally, an attached authentication tag T asserts the authenticity of both the plaintext and any optional (unencrypted) associated data A. Each encryption call requires a unique nonce N as an additional input to "randomize" the encryption. Corresponding to the CAESAR call, Isap maintains security no matter how the nonce N is chosen, as long as the same nonce is never used for encryption with the same secret key twice. In this section, we define the Isap authenticated cipher (Subsection 3.1) and its building blocks:

- IsapEnc, a cipher that computes the ciphertext C from the plaintext P and nonce N using the secret key K_E (Subsection 3.3).

- IsapMac, a message authentication code that computes the authentication tag T from the ciphertext C, associated data A, and nonce N using the secret key K_A (Subsection 3.2).

- IsapRk, a function used internally by IsapMac to absorb the secret key K_A (Subsection 3.2).

We propose to implement each building block with variants of the sponge construction using the same permutation size, but different round numbers and rates. We specify several recommended parameter sets in Subsection 3.4.

3.1 Authenticated Encryption Scheme

Isap is a family of sponge-based authenticated encryption schemes $\text{Isap}_{a,b,c}^{r_1,r_2,r_3}\text{-}k$, where the key size k defines the security level. Isap is an Encrypt-then-MAC design and uses two k-bit keys K_A and K_E ($K = K_A \| K_E$) for IsapMac and IsapEnc, respectively. The length of the tag T and nonce N is also k bits. Each family member is additionally parametrized by several parameters: different round numbers a, b, and c for the permutations and different rates r_1, r_2, and r_3.

The inputs for the authenticated encryption algorithm $\mathcal{E}$ are the secret key $K = K_A \| K_E$, the public nonce N, and associated data A and plaintext P of arbitrary length. Its outputs are the tag T and the ciphertext C with the exact same length as the plaintext P:

$$\mathcal{E}(K, N, A, P) = (C, T).$$

The inputs for the authenticated decryption algorithm $\mathcal{D}$ are the secret key $K = K_A \| K_E$, the public nonce N, the tag T, and associated data A and ciphertext C of arbitrary length. Its outputs are the plaintext P if the verification succeeds, or $\perp$ if the verification fails:

$$\mathcal{D}(K, N, A, C, T) \in \{P, \perp\}.$$

Isap is based on the well-established Encrypt-than-MAC paradigm. Hence, Isap is composed of an encryption algorithm $\text{IsapEnc}_{b,c}^{r_2,r_3}\text{-}k$ and a message authentication code $\text{IsapMac}_{a,b,c}^{r_1,r_2}\text{-}k$. The interaction between them is captured in Algorithm 1, where the authenticated encryption $\mathcal{E}$ and authenticated decryption $\mathcal{D}$ are specified.

Algorithm 1: Authenticated encryption and decryption procedures.

Auth. Encryption $\mathcal{E}(K, N, A, P)$	Auth. Decryption $\mathcal{D}(K, N, A, C, T)$
Input: key $K = K_A \| K_E$, $K_A \in \{0,1\}^k$, $K_E \in \{0,1\}^k$, Nonce $N \in \{0,1\}^k$, associated data $A \in \{0,1\}^*$, plaintext $P \in \{0,1\}^*$ **Output:** ciphertext $C \in \{0,1\}^*$, tag $T \in \{0,1\}^k$	**Input:** key $K = K_A \| K_E$, $K_A \in \{0,1\}^k$, $K_E \in \{0,1\}^k$, Nonce $N \in \{0,1\}^k$, associated data $A \in \{0,1\}^*$, ciphertext $C \in \{0,1\}^*$, Tag $T \in \{0,1\}^k$ **Output:** plaintext $P \in \{0,1\}^*$, or $\perp$
Encryption $C \leftarrow \text{IsapEnc}_{b,c}^{r_2,r_3}\text{-}k(K_E, N, P)$ **Authentication** $T \leftarrow \text{IsapMac}_{a,b,c}^{r_1,r_2}\text{-}k(K_A, N, A, C)$ **return** C, T	**Verification** $T' \leftarrow \text{IsapMac}_{a,b,c}^{r_1,r_2}\text{-}k(K_A, N, A, C)$ **if** $T \neq T'$ **return** $\perp$ **Decryption** $P \leftarrow \text{IsapEnc}_{b,c}^{r_2,r_3}\text{-}k(K_E, N, C)$ **return** P

3.2 Authentication Part

For our message authentication code IsapMac, we turn a sponge-based hash function into a suffix-MAC as shown in Figure 2. While the data is absorbed as in a sponge, we use a duplex-like approach to inject the secret key K_A. Here, the k-bit outer part of the state is processed together with the secret key K_A to derive a session key K_A^*, which is then further used as the outer part.

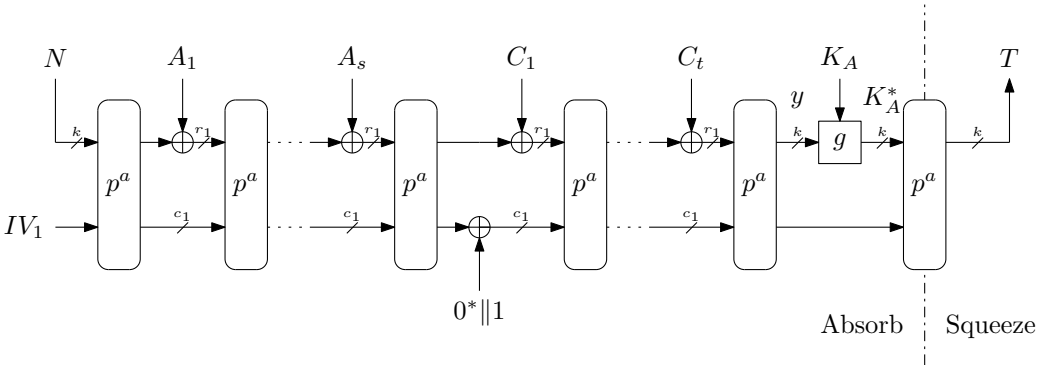


Figure 2: IsapMac used for authentication.

Alternatively, IsapMac can be seen as a sponge-based suffix-MAC, which uses a function g to absorb the secret key K_A instead of an XOR operation. Similar to fresh re-keying schemes [MSGR10], the sole purpose of g is to protect the static master key K_A against various classes of passive side-channel attacks, most prominently differential power analysis. Hence, we will subsequently call g our re-keying function. In our case, we will use IsapRk as re-keying function, which is shown in Figure 3.

IsapMac computes the tag T as follows. Both associated data A and ciphertext C are each padded using a 10^* padding to a length that is a multiple of the rate r_1. The internal state is initialized with the k-bit nonce N and a constant initial value IV_1. Then, s blocks of associated data $A_{1...s}$ and t blocks of ciphertext $C_{1...t}$ are absorbed using the a-round permutation p^a. Similar to Ascon [DEMS14], the XOR of a single bit '1' to the inner part of the state serves as domain separation between associated data and ciphertext. Note that a dedicated domain separation between nonce and associated data is not needed, since the nonce is of a fixed length of k bits. Finally, the key K_A is absorbed via g and the k-bit tag T is squeezed after a final call of the permutation.

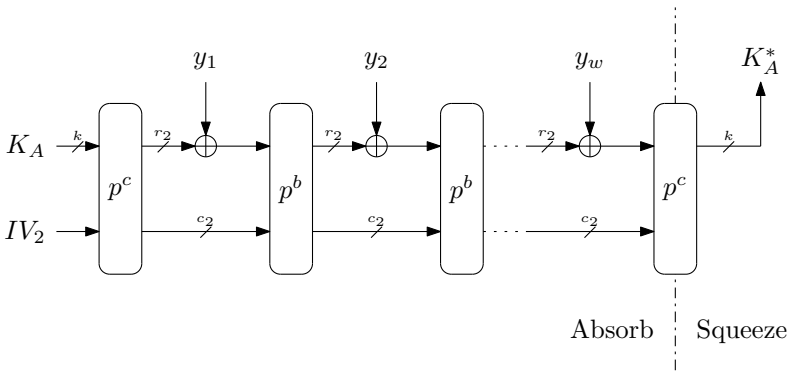


Figure 3: IsapRk used to process the master key K_A.

Instead of a plain XOR, the function $g(K_A, y) = K_A^*$ is used to absorb K_A. To evaluate g, its internal state is initialized with the key K_A and a constant IV_2, followed by an application of the c-round permutation p^c. The k-bit value y is absorbed using a rate size r_2 and the b-round permutation p^b. Finally, the output K_A^* is squeezed using a rate size k and the c-round permutation p^c. The details of IsapMac and IsapRk are also summarized in Algorithm 3 in the appendix. For verification, the tag T' is re-computed in the same way from the received nonce N, associated data A, and ciphertext C.

3.3 Encryption Part

To encrypt the plaintext, we use a sponge-based construction very similar to IsapRk (see Figure 4). We initialize the internal state with the secret key K_E and a constant IV_3, followed by an application of the c-round permutation p^c. The k-bit nonce N is absorbed using a rate of r_2 bits and the b-round permutation p^b. Then, we squeeze a keystream of the same length as the plaintext P using a rate of r_3 bits and the c-round permutation p^c. The ciphertext C is computed as the XOR of the plaintext P and the keystream.

For decryption, the same keystream is computed from the nonce N and XORed to the ciphertext C to obtain the plaintext P. The detailed procedures for encryption and decryption are also given in Algorithm 4 in the appendix.

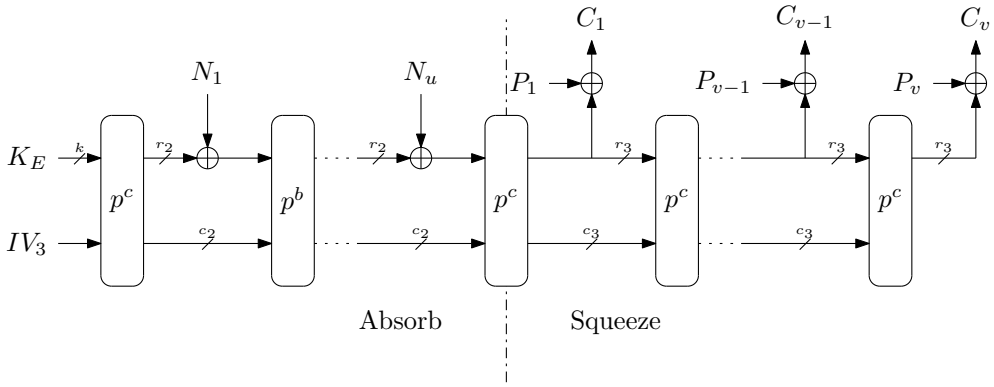


Figure 4: IsapEnc used for encryption.

3.4 Instantiations and Parameter Values

We propose to instantiate the required permutations p^a, p^b, and p^c from the 400-bit permutations Keccak-p[400,n_r], which are the application of the last n_r rounds of Keccak-f[400] [Nat15]. Hence, the only difference between p^a, p^b, and p^c is the different number of rounds a, b, and c that is used. A detailed specification of Keccak-p[400,n_r], including the state layout and specification of the inner and outer state parts, can be found in the submission document of the CAESAR candidate Keyak [BDP+14b].

Table 1 summarizes the recommended parameter sets for Isap. The first, Isap-128, is based on a conservative choice of the relevant parameters based on our design rationale and security analysis given in Section 4 and Section 5. Additionally, we also specify a more aggressive choice of parameters in Isap-128a to encourage further cryptanalysis as well as side-channel analysis. Both algorithms are designed to achieve 128 bits cryptographic security and practical security against side-channel attacks assuming an SPA-secure implementation.

The constant initial values IV_1, IV_2, IV_3, which serve as domain separation between the different algorithms, are specified in Table 2. They are defined as the concatenated bit

Table 1: Recommended parameter configurations for ISAP.

Name	Security level	Bit size of			Rounds		
	k	r_1	r_2	r_3	a	b	c
ISAP-128	128	144	1	144	20	12	12
ISAP-128a	128	144	1	144	16	1	8

values of the used parameter set, plus a different constant for each value, where each entry occupies 1 byte of space. The initial values are then padded with zeros until they reach the length of the permutation minus k bits. In the case of ISAP-128 and ISAP-128a, the IVs have a length of 272 bits, which is more than needed for the desired security level of 128 bits.

Table 2: Initial values for ISAP.

$\text{ISAP}_{a,b,c}^{r_1,r_2,r_3}$-$k$	IV_1	$1\|\|a\|\|b\|\|c\|\|r_1\|\|r_2\|\|r_3\|\|k\|\|0^*$
	IV_2	$2\|\|a\|\|b\|\|c\|\|r_1\|\|r_2\|\|r_3\|\|k\|\|0^*$
	IV_3	$3\|\|a\|\|b\|\|c\|\|r_1\|\|r_2\|\|r_3\|\|k\|\|0^*$
ISAP-128	IV_1	0x01140c0c90019080*
	IV_2	0x02140c0c90019080*
	IV_3	0x03140c0c90019080*
ISAP-128a	IV_1	0x0110010690019080*
	IV_2	0x0210010690019080*
	IV_3	0x0310010690019080*

4 Design Rationale

The main goal of ISAP is to provide security against passive side-channel attacks by design, while still providing good performance and a low hardware footprint. While mechanisms to counteract side-channel attacks and in particular DPA within the cipher itself (e.g., masking) lead to significant overheads and increase with the protection order, approaches based on fresh re-keying lead to much lower overheads. However, state-of-the-art schemes based on re-keying lack security against DPA in scenarios that require multiple decryption of the same input (with the same session key). ISAP is designed to be secure also in such scenarios.

4.1 An Authenticated Encryption Mode Secure Against DPA

For discussing the security of our scheme against differential power analysis (DPA), we prefer to give a more general, high-level view on our mode in Algorithm 2 to better extract the underlying idea. In contrast to the condensed and interwoven descriptions of IsapMac, IsapRk, and IsapEnc, the description in Algorithm 2 clearly shows the fresh re-keying roots of our scheme. Here, we essentially use the same assumptions and requirements as other re-keying schemes. Namely, we assume g_1, g_2 to be (DPA and SPA) secure re-keying functions and assume the implementations of ENC, DEC, and MAC to be secure against SPA attacks when processing arbitrarily long messages. However, there are no requirements on the implementation of the hash function H, since it processes only publicly known data.

To achieve security against DPA, our authenticated encryption mode in Algorithm 2 incorporates the re-keying approach discussed in Section 2 in an efficient Encrypt-then-MAC scheme. While simple re-keying of both a MAC and an encryption scheme can

Algorithm 2: Authenticated encryption and decryption procedures.

Auth. Encryption $\mathcal{E}(K, N, A, P)$	Auth. Decryption $\mathcal{D}(K, N, A, C, T)$
Input: key $K = K_A \| K_E$, $K_A \in \{0,1\}^k$, $K_E \in \{0,1\}^k$, Nonce $N \in \{0,1\}^k$, associated data $A \in \{0,1\}^*$, plaintext $P \in \{0,1\}^*$ **Output:** ciphertext $C \in \{0,1\}^*$, tag $T \in \{0,1\}^k$	**Input:** key $K = K_A \| K_E$, $K_A \in \{0,1\}^k$, $K_E \in \{0,1\}^k$, Nonce $N \in \{0,1\}^k$, associated data $A \in \{0,1\}^*$, ciphertext $C \in \{0,1\}^*$, Tag $T \in \{0,1\}^k$ **Output:** plaintext $P \in \{0,1\}^*$, or $\perp$
Encryption $K_E^* = g_1(N, K_E)$ $C = ENC_{N,K_E^*}(P)$ **Authentication** $y = H(N, A, C)$ $K_A^* = g_2(y, K_A)$ $T = MAC_{K_A^*}(y)$ **return** C, T	**Verification** $y = H(N, A, C)$ $K_A^* = g_2(y, K_A)$ $T' = MAC_{K_A^*}(y)$ **if** $T \neq T'$ **return** $\perp$ **Decryption** $K_E^* = g_1(N, K_E)$ $P = DEC_{N,K_E^*}(C)$ **return** P

only provide security for the encryption process, our scheme achieves side-channel security for multiple decryption as well. Namely, the verification guarantees the security of the decryption part in case of maliciously modified ciphertexts, while the *MAC* is protected by making its session key depend on the authenticated message itself. In the following, we give a detailed discussion on the DPA security of the two parts encryption/decryption and authentication/verification.

Encryption/Decryption. The encryption and decryption part is an instance of fresh-rekeying such as in [MSGR10, MPR+11]. Such schemes for fresh re-keying combine an SPA-secure encryption scheme *ENC* with a (DPA and SPA) secure re-keying function $g_1 : (K_E, N) \mapsto K_E^*$. As the nonce N that is used to derive the session key K_E^* must not be repeated, fresh session keys are guaranteed and DPA on the encryption scheme *ENC* is effectively prevented.

However, for decryption, there is the threat that an adversary could exploit multiple decryptions with the same session key K_E^* and induce a DPA setting within the decryption *DEC* by using different data, since multiple calls of *DEC* with the same nonce N are allowed. To prevent such a DPA scenario in our mode, verification is performed prior to decryption. Decrypting two different messages (associated data and ciphertext) with the same K_E^* indicates either a collision of g_1 for fixed K_E (depends on concrete instance of g_1, but usually negligible probability), or two ciphertexts that have been encrypted using the same nonce N. Since we require unique nonces, the latter implies that either the ciphertexts are identical, or one ciphertext has been forged. If a cryptographically secure MAC is used, the probability of a successful forgery is negligible and thus the tag verification will fail for one of the ciphertexts with overwhelming probability.

Authenticated ciphers require that no decrypted plaintext is released if tag verification fails. To ensure protection against DPA attacks, we go one step further and require a failed verification to abort the authenticated decryption process, so that the decryption part *DEC* never starts. This ensures that the same session key K_E^* is never used to decrypt distinct ciphertexts with *DEC*. Therefore, the verification is responsible for precluding DPA attacks on the decryption.

Authentication/Verification. The authentication/verification shown in Algorithm 2 is based on a hash-then-MAC paradigm. Here, a session key K_A^* is first derived via a

secure re-keying function g_2 from the hash value y that is computed from the nonce N, associated data A, and ciphertext C using a cryptographic hash function H. Then, a message authentication code (MAC) is used to compute the tag T from the hash value y and the session key K_A^*. This is similar to the construction of Pereira et al. [PSV15], who designed a leakage-resilient MAC based on the hash-then-MAC paradigm as well. However, the main difference to our approach is that in [PSV15], a random nonce N is used to derive the session key in the re-keying function. This, however, cannot provide protection against DPA for multiple verifications. Contrary to that, we use the hash of the message $y = H(N, A, C)$ to derive the session key K_A^* in order to securely allow multiple verifications while still providing protection against DPA.

In more detail, the MAC in Algorithm 2 computes the tag T using a different session key K_A^* for every distinct message (N, A, and C), because distinct messages result in distinct hash values in the absence of collisions. Hence, DPA on the MAC is prevented during the generation of the tag T as the same session key K_A^* is never used to authenticate distinct messages.

While the scheme by Pereira et al. [PSV15] also provides side-channel security during tag generation by the use of a unique nonce input N to the re-keying function, tag verification imposes different challenges. In fact, during tag verification one cannot rely on the uniqueness of the nonce anymore, because an attacker can usually modify the message (N, A, and C) to provoke multiple verifications with different data under the same nonce N and thus allowing for a DPA scenario. However, the MAC in Algorithm 2 prevents such a DPA scenario on the session key K_A^*, since K_A^* is bound to the data it processes. Namely, as y depends on the message (N, A, and C), the MAC session key $K_A^* = g(y, K_A)$ changes whenever the data changes. Adversaries cannot predictably influence y due to the use of a cryptographic hash function H. This guarantees that the key K_A^* is unique for every new message as long as there is neither a collision in the hash function H nor in the re-keying function g_2. Thus, DPA on the session key K_A^* is effectively prevented during verification.

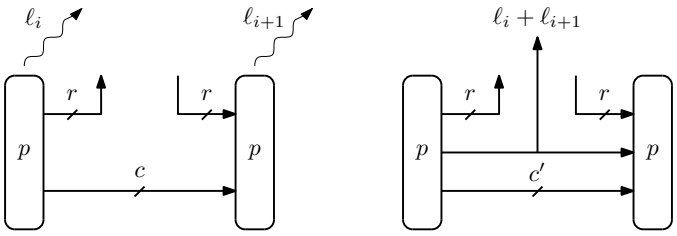
Note however that collisions in the re-keying function g_2 or the hash function H may result in the same session key K_A^* being used in MAC computations of different messages, thus allowing for a DPA. Yet, collisions in g_2 depend on the secret key K_A^* and therefore inputs causing collisions in g_2 cannot be calculated off-line. In contrast, collisions in the hash value y are directly observable and can be calculated off-line. The complexity of calculating collisions off-line is determined by the size of the hash. The generic complexity of finding a collision for an m-bit hash function is $2^{m/2}$. Hence, the size of the hash needs to be chosen depending on the potential threat of such an event, which depends on the concrete choice of functions for MAC and g_2.

4.2 Sponges and Side-Channels Leakage

While the mode of Subsection 4.1 ensures protection of the encryption, ENC, decryption DEC, and message authentication code MAC against DPA, the primitives implementing ENC, DEC, and MAC still have to withstand SPA attacks. Moreover, SPA protection is also mandatory for the implementations of g_1 and g_2, in addition to the requirement that they provide protection against DPA. Besides dedicated countermeasures like, e.g., shuffling, the order of the executed instructions, and already the choice of the used algorithms for encryption/decryption and MAC, play an important role for the resistance of the design against SPA.

Our choice for sponge-based designs is motivated by their suitability to model SPA leakage. Namely, the sponge parameters provide a convenient tool to argue on the side-channel security of keyed sponge constructions given bounded side-channel leakage of the single permutation.

For illustration, we model the leakage from a permutation p by allowing an adversary to learn a certain amount of the state between subsequent permutation calls as depicted in



(a) Leaking permutations (b) Sponge model

Figure 5: Leakage of information in sponge-based constructions.

Figure 5. Hereby, we use ℓ to denote the amount of information (in bits) that an attacker can learn about the state from the collected side-channel information. We do not care how and where the leakage is created within p, but let the adversary account the learned information to either the input or the output state of p. Therefore, given two consecutive permutations p with leakages ℓ_i and ℓ_{i+1}, respectively, the maximum an adversary might learn about the state is $\ell_i + \ell_{i+1}$. This means that if each leakage ℓ_i, ℓ_{i+1} is bounded by λ bits and the adversary can optimally combine these two leakages, the adversary will learn at most 2λ bits of the state between the respective two permutation calls.

The basic idea now is to use the sponge parameters to express a construction's capability to cope with the leakage generated by the permutation. In particular, the sponge parameters are adjusted according to the amount of information an adversary learned about the secret state. This means that if the adversary learns 2λ bits of the internal, secret state, the leaked bits can be considered as an increase of the rate, i.e., $r' = r + 2\lambda$, which results in a smaller capacity $c' = c - 2\lambda$ and thus reduced security. However, a reduced security level corresponding to a capacity of $c - 2\lambda$ bits is still guaranteed by the cryptographic properties of the permutation and the associated constrained-input constrained-output (CICO) problem [BDPV11a]. Sponge-based constructions can thus be considered to have bounded security loss for bounded leakage of the permutation.

Clearly, the challenge in practice is to build an implementation that bounds the leakage of p. Especially if many different types of devices have to use the same cryptographic algorithm it might be infeasible to make any realistic assumptions about the leakage of p. Nevertheless, the advantage of the sponge-based construction is that besides standard SPA countermeasures, like hiding and masking, the capacity is an additional and very natural security parameter that helps to increase the ability of a design to withstand side-channel attacks in practice.

While the above modelling and arguing about the leakage is quite useful, it points out a problem with the absorption of the key. If a key is directly absorbed, the upcoming permutation call directly leaks information about the key bits via side-channels. This has a direct effect on the security of the scheme if the used key length matches the security level. Hence, we propose to store the expanded key, after the application of p^c, for IsapRk and IsapEnc.

Besides giving a useful tool to model and argue about the SPA resistance, sponge-based constructions provide other significant advantages:

- The sponge construction is well-studied and has been analyzed and proven secure for different applications in a large amount of publications [JLM14, ADMV15, BDPV11b].
- It allows to implement a wide range of primitives (hash, MAC, cipher).
- Elegant and simple design, obvious state size, no key schedule, key is injected once.
- Little implementation overhead for decryption, since no inverse building blocks (permutation) are needed.

4.3 Design of IsapMac

To get more insight into the design rationals behind IsapMac, we first take a look at a direct instantiation of the authentication/verification described in Algorithm 2. Figure 6 sketches such an instantiation using a sponge-based hash function and suffix MAC. In contrast to the description in Algorithm 2, the MAC is computed directly using the data instead of the hash value. This leads to a construction where the data is processed twice.

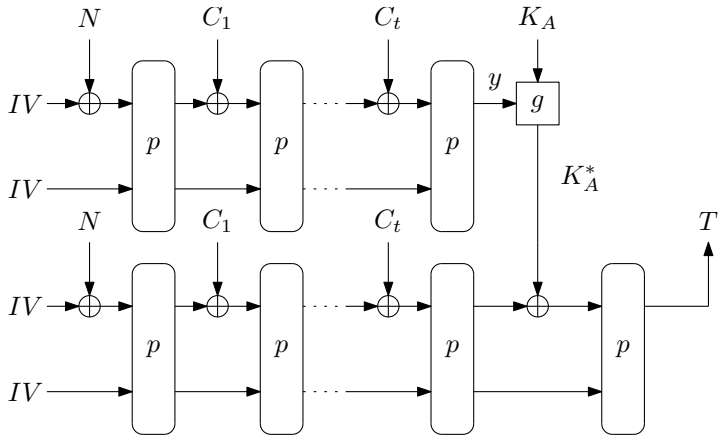


Figure 6: Sketch of authentication/verification using sponge-based hash and suffix MAC.

However, it is possible to omit the hash function and process the key in a manner that resembles the duplex construction [BDPV11b]. As shown in Figure 7, the outer part of the state is used to derive a session key K_A^*, which is then absorbed. This principle is further tweaked (e.g., by implicitly assuming that the employed re-keying function is $g(K_A, y) \oplus y$ to eliminate the XOR used to absorb K_A^*) which leads to IsapMac as presented in Subsection 3.2 (Figure 2). An alternative way of interpreting IsapMac is to see it as sponge-based suffix-MAC that uses a secure re-keying function g to absorb the secret key K_A instead of an XOR. Due to the simplicity of this description, we have chosen to stick to it throughout the paper.

Bertoni et al. [BDPV11a] showed that one can always turn a sponge into a MAC by either putting the key before (prefix-MAC) or after the message (suffix-MAC), as this always gives a pseudo-random function as long as the sponge itself behaves like a random oracle. Compared to a "standard" sponge-based suffix-MAC, IsapMac uses a secure re-keying function g to absorb the secret key K_A. While there are several options for g, e.g., the polynomial multiplication in [MSGR10], we use the function IsapRk as g.

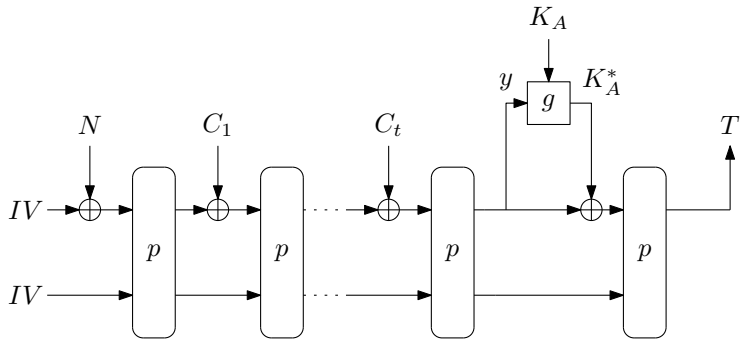


Figure 7: Sketch of authentication/verification just using a sponge-based suffix MAC.

Table 3: Complexity for receiving a v-collision for a 128-bit session key k_2.

v	2	3	4	5	$\cdots$	34
complexity	$2^{64.5}$	$2^{86.2}$	$2^{97.1}$	$2^{103.8}$	$\ldots$	2^{128}

Although IsapRk is not a permutation for a fixed key as, e.g., a polynomial multiplication, we do not expect any negative consequences on the security when absorbing the secret key via a function that ideally behaves like a pseudo-random function.

Instead of using a distinct padding or frame bits for domain separation between associated data and ciphertext, we follow the approach of Ascon [DEMS14] and XOR a single '1' to the inner part of the state. Although this reduces the capacity by one bit in the worst case, the practical security loss is considered to be negligible.

IsapMac prevents DPA on the tag computation in two ways. First, and as shown in Figure 2, the MAC session key K_A^* is derived from the hash value y and the MAC master key K_A via a secure re-keying function g, thus prohibiting DPA on K_A. Second, the design prevents DPA on the MAC session key K_A^* by binding it to the data being processed, thus leading to different MAC session keys K_A^* for different data.

As already mentioned before, a collision in y allows for two side-channel measurements of the MAC using different data but the same MAC session key K_A^*. This holds true for IsapMac as well. Yet, to perform a successful DPA, usually more than two traces will be needed to recover one fixed session key K_A^*. Such a setting occurs with hash multi-collisions. The generic complexity for finding a v-collision is $\sqrt[v]{v! \cdot 2^{m(v-1)}}$ [STKT06]. Luckily, the complexity is quite high already for small values of v as shown in Table 3 for a 128-bit value y.

However, we want to stress that even though a DPA attack exploiting multi-collisions might be able to recover the session key K_A^* of IsapMac, this does not imply a key recovery attack on the master key K_A, since our used re-keying function g (IsapRk) is hard to invert.

4.4 Design of IsapRk

The re-keying function IsapRk used in IsapMac is a sponge-based design as depicted in Figure 3. When setting the rate r_2 to 1, the design is related to the classical GGM construction [GGM86] and can be seen as their sponge-based equivalent, similar to [TS14]. The basic idea in IsapRk is to make DPA infeasible by reducing the input data complexity accordingly. For this purpose, a secret state is constantly updated with small portions of of public data by repeating two phases, (1) modifying the secret state according to the public data, and (2) updating the state such that predictions on the future state based on the absorbed public data become infeasible.

Sponge-based constructions are an ideal choice to implement this basic idea as the rate directly influences the input data complexity for each permutation. IsapRk follows this approach and first initializes the internal state by applying the initial permutation p^c to the padded master key K_A. Then, IsapRk repeatedly injects r_2 nonce bits into the state, each separated by a permutation call p^b. After full absorption of the nonce and finalization using p^c, the session key K_A^* is output. This working principle is similar to sponge instances of a prefix-MAC. While for general MAC computations the absorption rate can be as big as the state size [BDPV12], IsapRk uses a small absorption rate $r_2 = 1$ to limit the data complexity exploitable in a DPA.

In terms of DPA security, a permutation p^b will produce the leakages for two different public inputs, thus IsapRk is 2-limiting per permutation call. This results in IsapRk being a secure re-keying function (regarding DPA) under the assumption that the combined leakage resulting from the processing of two different public inputs is bounded such that

DPA on the secret state is infeasible. This is a common assumption also used in recent block-cipher based instantiations of the GGM construction by Faust et al. [FPS12] or the 2PRG primitive by Standaert et al. [SPY+10]. The reason for using a different permutation p^c at the beginning of IsapRk lies in the fact that some of the concrete instances of IsapRk use a small number of rounds for p^b compared to p^c and we want to ensure good diffusion of the key bits across the whole state before the first non-secret bits are absorbed.

4.5 Design of IsapEnc

The encryption algorithm IsapEnc is an instance of fresh re-keying [MSGR10, MPR+11] that combines the secure re-keying function IsapRk in the initialization phase with a sponge-based stream cipher in the processing phase. However, for the analysis it is more natural to see it as an extension of IsapRk with a longer squeezing phase to produce a keystream of arbitrary length.

As the initialization part is equivalent to IsapRk, it is secure against passive side-channel attacks in consideration of the same aspects, i.e., a small rate $r_2 = 1$ to inject the nonce N with low data complexity. To obtain cryptographic security on the processing part of IsapEnc, the nonce N must not be repeated for different plaintexts. This guarantees that the key stream is unpredictable and unique for different encryptions. As a consequence, DPA on the encryption itself is prevented as well. Moreover, as a part of the authenticated encryption scheme Isap, IsapEnc remains secure against DPA also for multiple decryption of the same data, since it is guaranteed that this data is always decrypted under the same nonce. As mentioned before, current schemes lack this functionality and become vulnerable to DPA if an attacker tampers with the ciphertext or nonce. In IsapEnc, such attack becomes infeasible by using the generic composition Encrypt-then-MAC, i.e., performing verification prior to decryption. Namely, the authentication part aborts the process if tag verification fails, which ensures that the same key is never used to decrypt distinct ciphertexts. Hence, the authentication part precludes DPA attacks on the decryption part.

4.6 Choice of the Permutation

In the case of sponge-based constructions, minimal suitable bit-sizes for permutations are tightly coupled with the aimed security level. Both instances Isap-128 and Isap-128a target 128-bit security. Hence, the capacity of IsapMac should be at least 256 bits, since it is a sponge-based suffix MAC and thus, we have to rely on the results of Bertoni et al. [BDPV08]. If we want to output the tag with one permutation call, while still retaining 256 bits for the capacity, this implies a minimal permutation size of 384 bits. Since Isap-128 and Isap-128a are also aimed for lightweight and low-cost applications, while high performance applications are not the main target, we do not want to increase the rate much and hence want to stay close to 384 bits. However, there is a lack of well-analyzed 384-bit permutations. Thus, we opted to use the well established and analyzed Keccak-p[400,n_r] permutations [Nat15].

Parameters for IsapMac. Since we aim for 128-bit security, we use IsapMac for both instances with a capacity c_1 of 256 bits, while allowing the remaining 144 bits as rate r_1. For the conservative choice Isap-128, we choose p^a to be the permutation Keccak-f[400] (Keccak-p[400,20]) that has 20 rounds as specified in the the Keccak SHA-3 submission (Version 3.0) [BDPV11c]. Since Keccak is the winner of the SHA-3 competition, its variants have been well analyzed. However, current attacks are far away from threatening full-round versions of Keccak. Therefore, we use for our aggressive variant Isap-128a the initial Keccak-p[400,16] with 16 rounds as proposed in the Keccak sponge function family main document (Version 1.2) [BDPV09].

Parameters for IsapRk and IsapEnc. Both IsapRk and IsapEnc are keyed sponge-based constructions with clearly separated absorbing and squeezing phases. According to recent results [BDPV12, GPT15, MRV15], we could set the capacity during the absorbing phase to $c_2 = 0$ and the capacity during the squeezing phase to a minimum of $c_3 = 128$ bits. However, we also have to bear side-channel attacks in mind. Hence, we set the rate to $r_1 = 1$, making the scheme essentially 2-limiting per permutation call p^b, while setting the rate $r_3 = 144$ bits to match the block size of IsapMac. In terms of our arguments of Subsection 4.2, this means that an attacker has to learn about 136 bits of information during invocations of p^b and about 64 bits of information via side-channels during the invocation of p^c, before the attacker is able to invert the sponge with a complexity less than 2^{128} to recover the secret key.

For the number of rounds for Isap-128, the CAESAR candidate Keyak serves as orientation. Hence, we use Keccak-p[400,12] for p^b and p^c. As for Keyak, we expect 12 rounds to be enough to create an unpredictable key-stream during the squeezing phase. Moreover, 12 rounds provide a clear separation between the single-bit injections during the absorption, so that partially known/leaked information about the internal secret state cannot be combined over one permutation call.

The CAESAR candidate Ketje serves as inspiration for the aggressive version Isap-128a. Similar to Ketje, only one round separates the absorption of the one bit elements using Keccak-p[400,1] for p^b. Note that here the side-channel leakage between single permutation calls can clearly be combined. For the squeezing phase, we orient the number of rounds on the "stride" permutation call of Ketje Sr, which has 6 rounds. However, in contrast to Ketje Sr, we have a higher rate of 144 bits during the squeezing phase. Hence, we have decided to add an additional security margin of 2 rounds and use the 8 round permutation Keccak-p[400,8] for p^c.

5 Security Analysis

Due to the prominence of Keccak [BDPV11c] as winner of the SHA-3 competition [Nat12], and Keyak [BDP+14b] and Ketje [BDP+14a] as submissions to CAESAR [CAE14], a plethora of cryptanalytic results for keyed and unkeyed sponge and duplex constructions using round reduced versions of the Keccak-f permutations, as well as on the permutations exist. While arguably the majority of the analyses focuses on the 1600-bit variant of the Keccak-f permutation, the similarity in structure of the permutation usually allows to apply the same techniques on smaller permutation variants. A good overview on existing analysis results on Keccak can be found in [JN15]. In this section, we recapitulate the from our point of view most relevant attacks on Keccak and discuss the applicability to our schemes. Finally, we conclude this section with a note on the side-channel security of Isap.

5.1 Permutation

Zero-sum distinguishers [AM09, BC10] are the permutation distinguishers penetrating the highest number of rounds. They exploit the low algebraic degree of the Keccak-f permutations creating sets of inputs and outputs, which sum to zero. Guo et al. [GLS16] present zero-sum distinguishers for 12 rounds of Keccak-f[1600] with a complexity of 2^{65} using a 3-round linear structure in the middle of the permutation, while achieving 2^{82} using a 2-round linear structure. They also claim for the 12-round 400-bit permutation Keccak-p[400,12] zero-sum distinguishers with a complexity 2^{82} using a 2-round linear structure, while 3-round structures seem to be inapplicable. However, to mount an attack using zero-sum distinguishers on sponges, an attacker would have to be able to choose inputs in the middle of the permutation. Thus, no attacks on Keyak and Ketje with

the 12-round KECCAK-p permutations are known that exploit zero-sum distinguishers. Therefore, we conclude that the same is true for ISAP-128, which also uses 12 rounds for ISAPENC and ISAPRK.

5.2 IsapRk and IsapEnc

ISAPRK and ISAPENC are sponge-based constructions where the secret key is injected during the beginning of the absorption phase, similar to a KECCAK prefix-MAC, KEYAK, or KETJE. We refer to such constructions as keyed sponges. The attacks penetrating the highest number of rounds for keyed sponges exploit the low algebraic degree of the KECCAK-f permutations. This includes the cube-like attacks by Dinur et al. [DMP+15], who present amongst others a keystream prediction for a KECCAK-based stream cipher which uses 9 rounds of the 1600-bit permutation to achieve 512-bit security with time complexity 2^{256}. Huang et al. [HWX+17] present conditional cube attacks, including a key-recovery attack on 8 rounds of KEYAK with a time complexity of 2^{74}.

In the case of ISAP-128, two factors prohibit those attacks. First of all, the permutation has 12 rounds, whereas the attacks are only capable of penetrating at most 9 rounds. Second, the nonce N or the hash value y are absorbed bitwise separated by 12 rounds of the permutation, which significantly reduces the ability of an attacker to exploit cubes in the first place. For ISAP-128a, the number of rounds between the bitwise injections of the nonce N or the hash value y is reduced to one. Still, this means having at least 128 rounds from the point where the key is introduced up to the point when a part of the state is leaked. Hence, we expect that conditional cube and cube-like attacks do not work on ISAP-128a.

Another important attack vector are linear and differential attacks. These are especially relevant in the case of ISAP-128a, where only the 1-round permutation is used for absorption and the 8-round permutation is used for squeezing the sponge. While having, e.g., colliding differential trails during absorption would also imply problems for KETJE, the situation changes for the squeezing phase. Due to the increased rate used in ISAP-128a compared to KETJE, an attacker has more freedom. For this reason, we have increased the number of rounds to 8 for p^c.

5.3 IsapMac

Since ISAPMAC is a suffix-MAC, attacks when unkeyed sponges are used as hash functions are also of concern. For instance, collision attacks on the hashing part of ISAPMAC have the potential to allow for forgeries. For KECCAK, collision attacks for up to 5 rounds were proposed by Dinur et al. [DDS13]. Recently, the 5-round challenges for 1600-bit and 800-bit permutations of the KECCAK crunchy crypto collision contest [BDPV14] have been solved, while the 5-round challenge for the 400-bit permutation is still open. Regarding pre-image attacks, attacks for up to 4 rounds for variants of KECCAK exist [MPS13, GLS16]. Taking these results together with the result for keyed sponges of Subsection 5.2, we conclude that having 20 rounds in the case of ISAP-128 and even 16 rounds in the case of ISAP-128a provide a sufficient security margin for ISAPMAC.

5.4 On the Side-Channel Security of Isap

While ISAP has been designed to be secure against DPA attacks, care has do be taken regarding SPA attacks. Although the single components ISAPMAC, ISAPRK and ISAPENC of ISAP have been designed keeping their resistance against SPA attacks in mind, additional countermeasures on implementation level for all components might be needed. In particular for the decryption, where several measurements for the same data are possible, dedicated countermeasures against SPA attacks are crucial.

As already pointed out by Medwed et al. [MSJ12], the concrete security of a construction against side-channel attacks highly depends on the way it is implemented and on the platform on which it is executed. For instance, they show that an implementation of the GGM construction using AES-128 on an 8-bit microcontroller can be broken by using template attacks. By making assumptions on the implementation, e.g., parallel execution of the S-boxes, Medwed et al. [MSJ12] and follow-up work [MSNF16] are able to provide security guarantees with respect to side-channel attacks for their constructions. In contrast, in this work we do not make any assumption on the way Isap is implemented and on the countermeasures used to protect the implementations. Clearly, an 8-bit microcontroller implementation needs more sophisticated SPA countermeasures than a parallel implementation of the round function. We consider the evaluation of the SPA resistance of various implementation strategies for Isap to be an interesting topic for further research.

6 Implementation

We implemented our authenticated encryption scheme Isap in the two configurations Isap-128 and Isap-128a as presented in Table 1. The actual implementation of both configurations is the same except for the number of rounds. The implementations employ a single instance of the 400-bit Keccak permutation that performs one round per cycle. The number of rounds performed is chosen at runtime depending on the executed algorithm, i.e., IsapEnc, IsapMac, or IsapRk. The synthesis results using a 130 nm UMC technology are shown in Table 4. The choice of 130 nm UMC technology is motivated by the tools which are available to us.

Table 4: Implementation of the AE modes (130 nm).

Function	Area [kGE]	Frequency [MHz]	Initialization [cycles]	Initialization [μs]	Runtime per Block [cycles]	Runtime per Block [μs]
Isap-128	14.0	169	3 401	20.1	36	0.20
Isap-128a	14.0	169	564	3.3	28	0.16

Area. As Isap-128 and Isap-128a use the same implementation design, they each consume 14.0 kGE of chip area. Most of the chip area is due to the Keccak core, which consumes 8.3 kGE. The remaining logic is required for multiplexing and a temporary state register to hold the hash value within IsapMac when performing the secure re-keying function IsapRk. A sole implementation of the secure re-keying function IsapRk yields roughly the same size as the Keccak core itself and is thus slightly smaller than other re-keying functions like a masked polynomial multiplication [MSGR10] or an implementation of the GGM tree using an AES core computing 1 round per cycle [SPY+10].

Runtime. The measured runtime is broken down into two parts: the time for performing initialization, and the time for encrypting and authenticating a 144-bit message block. The runtime of performing initialization is dominated by performing the re-keying operations in both IsapEnc and IsapMac and is independent of the length of the message. Its impact on runtime thus vanishes for long messages. The runtime for processing a single 144-bit block is also independent of the length of the message, but strongly influences the overall runtime for long messages.

Compared to the conservative parameterization Isap-128, the more aggressive parameters of Isap-128a yield a speed-up of 83 % for initialization and 22 % for the processing of a message block. While the very high speed-up during initialization is highly beneficial

for short messages, the speed-up observed for encryption and authentication of a 144-bit message block dominates for long messages.

Comparison. ISAP is an efficient authenticated encryption scheme with low hardware footprint that prevents DPA by design. While ISAP is based on a standard implementation of the 400-bit KECCAK permutation and thus only adds a little hardware overhead, a first-order secure threshold implementation increases the area by a factor of 3–4 [BDN+13]. Similar for AES the area for first-order secure masked implementations [DRB+16, GMK17] increases accordingly. When higher-order DPA security is required, the hardware overhead of masking rises even more [GMK17]. Consequently, the implementation cost of standard authenticated encryption modes for AES such as AES-CCM and AES-GCM secured via masking rises accordingly.

7 Conclusion and Open Questions

While current authenticated encryption schemes such as the CAESAR candidates ASCON, KETJE/KEYAK, PRIMATES, and SCREAM are designed to reduce the overhead of side-channel countermeasures like masking on an implementation level, we explored in this work how side-channel attacks can be tackled on an algorithmic level, while still fulfilling the functional requirements of the CAESAR call. Probably the most notable resulting restriction of this is that it is not possible to make any assumptions on the choice of the nonce, besides the fact that the nonce has to be unique per encryption (e.g. it must be possible to implement the nonce as simple counter on encryption side). Hence, the decrypting/verification unit has no influence on the choice of the nonce and thus has to allow multiple decryptions/verifications of (different) ciphertexts with the same nonce.

As a result, we proposed ISAP, an authenticated encryption scheme that incorporates ideas from fresh re-keying to withstand DPA attacks. In contrast to existing fresh re-keying schemes, ISAP protects the decryption/verification unit against DPA attacks, although the decryption/verification unit does not contribute to the nonce that is used for encryption. This feature does not only reduce communication overhead, but it enables several use cases that are not feasible with current re-keying schemes such as simply storing encrypted data and decrypting it later multiple times. The results of our hardware implementation show that the concrete instances ISAP-128 and ISAP-128a can be implemented in a straightforward manner with an area of only 14 kGE, while offering security against DPA attacks even for multiple decryption. Therefore, we think that ISAP is a valuable addition to the existing pool of symmetric authenticated encryptions schemes and hope that its novel underlying ideas and concepts will stimulate discussion and trigger future work in this direction.

Acknowledgments

The authors would like to thank Mario Werner for many helpful discussions and providing his hardware description of KECCAK.

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644052 (HECTOR) and agreement No 681402 (SOPHIA).

Furthermore, this work has been supported in part by the Austrian Research Promotion Agency (FFG) under grant number 845589 (SCALAS) and by the Austrian Science Fund (project P26494-N15).

References

[ABB+14] Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Florian Mendel, Bart Mennink, Nicky Mouha, Qingju Wang, and Kan Yasuda. PRIMATEs. Submission to the CAESAR competition: http://competitions.cr.yp.to, 2014.

[ADMV15] Elena Andreeva, Joan Daemen, Bart Mennink, and Gilles Van Assche. Security of keyed sponge constructions using a modular proof approach. In Gregor Leander, editor, *FSE 2015*, volume 9054 of *LNCS*, pages 364–384. Springer, 2015.

[AM09] Jean-Philippe Aumasson and Willi Meier. Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. https://131002.net/data/papers/AM09.pdf, 2009.

[BC10] Christina Boura and Anne Canteaut. A zero-sum property for the KECCAK-f permutation with 18 rounds. In *ISIT 2010*, pages 2488–2492. IEEE, 2010.

[BCF+15] Sonia Belaïd, Jean-Sébastien Coron, Pierre-Alain Fouque, Benoît Gérard, Jean-Gabriel Kammerer, and Emmanuel Prouff. Improved side-channel analysis of finite-field multiplication. In Tim Güneysu and Helena Handschuh, editors, *CHES 2015*, volume 9293 of *LNCS*, pages 395–415. Springer, 2015.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, 2004.

[BDH+14] Sonia Belaïd, Fabrizio De Santis, Johann Heyszl, Stefan Mangard, Marcel Medwed, Jörn-Marc Schmidt, François-Xavier Standaert, and Stefan Tillich. Towards fresh re-keying with leakage-resilient PRFs: Cipher design principles and analysis. *J. Cryptographic Engineering*, 4(3):157–171, 2014.

[BDN+13] Begül Bilgin, Joan Daemen, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Gilles Van Assche. Efficient and first-order DPA resistant implementations of Keccak. In Aurélien Francillon and Pankaj Rohatgi, editors, *CARDIS 2013*, volume 8419 of *LNCS*, pages 187–199. Springer, 2013.

[BDP+14a] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Ketje. Submission to the CAESAR competition: http://competitions.cr.yp.to, 2014.

[BDP+14b] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Keyak. Submission to the CAESAR competition: http://competitions.cr.yp.to, 2014.

[BDPV08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 181–197. Springer, 2008.

[BDPV09] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Keccak sponge function family main document (Version 1.2). http://keccak.noekeon.org/Keccak-main-1.2.pdf, 2009.

[BDPV11a] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Cryptographic sponge functions (Version 0.1). http://sponge.noekeon.org/, 2011.

[BDPV11b] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In Ali Miri and Serge Vaudenay, editors, *SAC 2011*, volume 7118 of *LNCS*, pages 320–337. Springer, 2011.

[BDPV11c] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. The Keccak SHA-3 submission (Version 3.0). http://keccak.noekeon.org/Keccak-submission-3.pdf, 2011.

[BDPV12] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Permutation-based Encryption, Authentication and Authenticated Encryption. DIAC Workshop Record (http://www.hyperelliptic.org/djb/diac/record.pdf), 2012.

[BDPV14] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak crunchy crypto collision and pre-image contest. http://keccak.noekeon.org/crunchy_contest.html, 2014.

[BFG14] Sonia Belaïd, Pierre-Alain Fouque, and Benoît Gérard. Side-channel analysis of multiplications in GF(2^{128}) – Application to AES-GCM. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014*, volume 8874 of *LNCS*, pages 306–325. Springer, 2014.

[BKP+16] Francesco Berti, François Koeune, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Leakage-resilient and misuse-resistant authenticated encryption. Cryptology ePrint Archive, Report 2016/996, 2016. http://eprint.iacr.org/2016/996.

[CAE14] CAESAR committee. CAESAR: Competition for authenticated encryption: Security, applicability, and robustness. http://competitions.cr.yp.to/, 2014.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, 2002.

[DDS13] Itai Dinur, Orr Dunkelman, and Adi Shamir. Collision attacks on up to 5 rounds of SHA-3 using generalized internal differentials. In Shiho Moriai, editor, *FSE 2013*, volume 8424 of *LNCS*, pages 219–240. Springer, 2013.

[DEMS14] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon. Submission to the CAESAR competition: http://competitions.cr.yp.to, 2014.

[DFH+16] Stefan Dziembowski, Sebastian Faust, Gottfried Herold, Anthony Journault, Daniel Masny, and François-Xavier Standaert. Towards sound fresh re-keying with hard (physical) learning problems. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016*, volume 9815 of *LNCS*, pages 272–301. Springer, 2016.

[DKM+15] Christoph Dobraunig, François Koeune, Stefan Mangard, Florian Mendel, and François-Xavier Standaert. Towards fresh and hybrid re-keying schemes with beyond birthday security. In Naofumi Homma and Marcel Medwed, editors, *CARDIS 2015*, volume 9514 of *LNCS*, pages 225–241. Springer, 2015.

[DMP+15] Itai Dinur, Pawel Morawiecki, Josef Pieprzyk, Marian Srebrny, and Michal Straus. Cube attacks and cube-attack-like cryptanalysis on the round-reduced Keccak sponge function. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 733–761. Springer, 2015.

[DRB+16] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking AES with $d + 1$ shares in hardware. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 194–212. Springer, 2016.

[FPS12] Sebastian Faust, Krzysztof Pietrzak, and Joachim Schipper. Practical leakage-resilient symmetric cryptography. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 213–232. Springer, 2012.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[GJ16] Qian Guo and Thomas Johansson. A new birthday-type algorithm for attacking the fresh re-keying countermeasure. Cryptology ePrint Archive, Report 2016/225, 2016. http://eprint.iacr.org/2016/225.

[GLS+14] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, Kerem Varici, François Durvaux, Lubos Gaspar, and Stéphanie Kerckhoff. SCREAM. Submission to the CAESAR competition: http://competitions.cr.yp.to, 2014.

[GLS16] Jian Guo, Meicheng Liu, and Ling Song. Linear structures: Applications to cryptanalysis of round-reduced Keccak. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016*, volume 10031 of *LNCS*, pages 249–274, 2016.

[GMK17] Hannes Gross, Stefan Mangard, and Thomas Korak. An efficient side-channel protected aes implementation with arbitrary protection order. In Helena Handschuh, editor, *CT-RSA 2017*, volume 10159 of *LNCS*, pages 95–112. Springer, 2017.

[GPT15] Peter Gazi, Krzysztof Pietrzak, and Stefano Tessaro. The exact PRF security of truncation: Tight bounds for keyed sponges and truncated CBC. In Rosario Gennaro and Matthew Robshaw, editors, *CRYPTO 2015*, volume 9215 of *LNCS*, pages 368–387. Springer, 2015.

[HWX+17] Senyang Huang, Xiaoyun Wang, Guangwu Xu, Meiqin Wang, and Jingyuan Zhao. Conditional cube attack on reduced-round Keccak sponge function. In *EUROCRYPT 2017*, 2017. (to appear).

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, 2003.

[JLM14] Philipp Jovanovic, Atul Luykx, and Bart Mennink. Beyond $2^{c/2}$ security in sponge-based authenticated encryption modes. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014*, volume 8873 of *LNCS*, pages 85–104. Springer, 2014.

[JN15] Jérémy Jean and Ivica Nikolic. Internal differential boomerangs: Practical analysis of the round-reduced Keccak-f permutation. In Gregor Leander, editor, *FSE 2015*, volume 9054 of *LNCS*, pages 537–556. Springer, 2015.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO '99*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.

[Koc03] Paul Kocher. Leak Resistant Cryptographic Indexed Key Update, US Patent 6539092, 2003.

[MBKP11] Amir Moradi, Alessandro Barenghi, Timo Kasper, and Christof Paar. On the vulnerability of FPGA bitstream encryption against power analysis attacks: extracting keys from Xilinx Virtex-II FPGAs. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *CCS 2011*, pages 111–124. ACM, 2011.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks – Revealing the secrets of smart cards*. Springer, 2007.

[MPR+11] Marcel Medwed, Christophe Petit, Francesco Regazzoni, Mathieu Renauld, and François-Xavier Standaert. Fresh re-keying II: Securing multiple parties against side-channel and fault attacks. In Emmanuel Prouff, editor, *CARDIS 2011*, volume 7079 of *LNCS*, pages 115–132. Springer, 2011.

[MPS13] Pawel Morawiecki, Josef Pieprzyk, and Marian Srebrny. Rotational cryptanalysis of round-reduced Keccak. In Shiho Moriai, editor, *FSE 2013*, volume 8424 of *LNCS*, pages 241–262. Springer, 2013.

[MRV15] Bart Mennink, Reza Reyhanitabar, and Damian Vizár. Security of full-state keyed sponge and duplex: Applications to authenticated encryption. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015*, volume 9453 of *LNCS*, pages 465–489. Springer, 2015.

[MS16] Amir Moradi and Tobias Schneider. Improved side-channel analysis attacks on Xilinx bitstream encryption of 5, 6, and 7 series. In François-Xavier Standaert and Elisabeth Oswald, editors, *COSADE 2016*, volume 9689 of *LNCS*, pages 71–87. Springer, 2016.

[MSGR10] Marcel Medwed, François-Xavier Standaert, Johann Großschädl, and Francesco Regazzoni. Fresh re-keying: Security against side-channel and fault attacks for low-cost devices. In Daniel J. Bernstein and Tanja Lange, editors, *AFRICACRYPT 2010*, volume 6055 of *LNCS*, pages 279–296. Springer, 2010.

[MSJ12] Marcel Medwed, François-Xavier Standaert, and Antoine Joux. Towards super-exponential side-channel security with efficient leakage-resilient PRFs. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 193–212. Springer, 2012.

[MSNF16] Marcel Medwed, François-Xavier Standaert, Ventzislav Nikov, and Martin Feldhofer. Unknown-input attacks in the parallel setting: Improving the security of the CHES 2012 leakage-resilient PRF. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016*, volume 10031 of *LNCS*, pages 602–623, 2016.

[Nat12] National Institute of Standards and Technology. SHA-3 competition. http://csrc.nist.gov/groups/ST/hash/sha-3/index.html, 2007–2012.

[Nat15] National Institute of Standards and Technology. FIPS PUB 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Federal Information Processing Standards Publication 202, U.S. Department of Commerce, August 2015.

[OC14] Colin O'Flynn and Zhizhang (David) Chen. ChipWhisperer: An open-source platform for hardware embedded security research. In Emmanuel Prouff, editor, *COSADE 2014*, volume 8622 of *LNCS*, pages 243–260. Springer, 2014.

[Pie09] Krzysztof Pietrzak. A leakage-resilient mode of operation. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 462–482. Springer, 2009.

[PM16] Peter Pessl and Stefan Mangard. Enhancing side-channel analysis of binary-field multiplication with bit reliability. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 255–270. Springer, 2016.

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 142–159. Springer, 2013.

[PSV15] Olivier Pereira, François-Xavier Standaert, and Srinivas Vivek. Leakage-resilient authentication and encryption from symmetric cryptographic primitives. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 96–108. ACM, 2015.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): measures and counter-measures for smart cards. In Isabelle Attali and Thomas P. Jensen, editors, *E-smart 2001*, volume 2140 of *LNCS*, pages 200–210. Springer, 2001.

[ROSW16] Eyal Ronen, Colin O'Flynn, Adi Shamir, and Achi-Or Weingarten. IoT goes nuclear: Creating a ZigBee chain reaction. Cryptology ePrint Archive, Report 2016/1047, 2016. http://eprint.iacr.org/2016/1047.

[SPY+10] François-Xavier Standaert, Olivier Pereira, Yu Yu, Jean-Jacques Quisquater, Moti Yung, and Elisabeth Oswald. Leakage resilient cryptography in practice. In Ahmad-Reza Sadeghi and David Naccache, editors, *Towards Hardware-Intrinsic Security – Foundations and Practice*, Information Security and Cryptography, pages 99–134. Springer, 2010.

[STKT06] Kazuhiro Suzuki, Dongvu Tonien, Kaoru Kurosawa, and Koji Toyota. Birthday paradox for multi-collisions. In Min Surp Rhee and Byoungcheon Lee, editors, *ICISC 2006*, volume 4296 of *LNCS*, pages 29–40. Springer, 2006.

[TS14] Mostafa M. I. Taha and Patrick Schaumont. Side-channel countermeasure for SHA-3 at almost-zero area overhead. In *HOST 2014*, pages 93–96. IEEE Computer Society, 2014.

[TS15] Mostafa M. I. Taha and Patrick Schaumont. Key updating for leakage resiliency with application to AES modes of operation. *IEEE Trans. Information Forensics and Security*, 10(3):519–528, 2015.

[UWM17] Thomas Unterluggauer, Mario Werner, and Stefan Mangard. Side-channel plaintext-recovery attacks on leakage-resilient encryption. In *DATE 2017*, 2017. (to appear).

A Algorithms

Algorithm 3: Suffix MAC IsapMac and re-keying function IsapRk.

$\textsc{IsapMac}^{r_1,r_2}_{a,b,c}$-$k$ $(K_A,\ N,\ A,\ C)$	$\textsc{IsapRk}^{r_2}_{b,c}$-$k(K_A, y)$
Input: key $K_A \in \{0,1\}^k$, nonce $N \in \{0,1\}^k$, associated data $A \in \{0,1\}^*$, ciphertext $C \in \{0,1\}^*$ **Output:** tag $T \in \{0,1\}^k$	**Input:** key $K_A \in \{0,1\}^k$, $y \in \{0,1\}^*$ **Output:** sessionkey $K_A^* \in \{0,1\}^k$

$\ell = \lvert A\rvert \mod r_1$
$A_1 \dots A_s \leftarrow r_1$-bit blocks of $A \,\|\, 1 \,\|\, 0^{r_1-1-\ell}$
$\ell = \lvert C\rvert \mod r_1$
$C_1 \dots C_t \leftarrow r_1$-bit blocks of $C \,\|\, 1 \,\|\, 0^{r_1-1-\ell}$
$S \leftarrow N \,\|\, IV_1$
$S \leftarrow p^a(S)$
Absorbing Associated Data
for $i = 1, \dots, s$ **do**
$S \leftarrow p^a((S_{r_1} \oplus A_i) \,\|\, S_{c_1})$
$S \leftarrow S \oplus (0^{r_1+c_1-1} \,\|\, 1)$
Absorbing Ciphertext
for $i = 1, \dots, t$ **do**
$S \leftarrow p^a((S_{r_1} \oplus C_i) \,\|\, S_{c_1})$
Squeezing Tag
$K_A^* \leftarrow \textsc{IsapRk}^{r_2}_{b,c}$-$k(K_A, \lceil S \rceil^k)$
$S \leftarrow p^a(K_A^* \,\|\, \lfloor S \rfloor_{r_1+c_1-k})$
$T \leftarrow \lceil S \rceil^k$
return T
	$\ell = \lvert y\rvert \mod r_2$
	if $\ell = 0$ **then**
	$y_1 \dots y_w \leftarrow r_2$-bit blocks of y
	else
	$y_1 \dots y_w \leftarrow r_2$-bit blocks of $y \,\|\, 0^{r_2-\ell}$
	$S \leftarrow K_A \,\|\, IV_2$
	Absorb
	$S \leftarrow p^c(S)$
	$S \leftarrow (S_{r_2} \oplus y_1) \,\|\, S_{c_2}$
	for $i = 2, \dots, w$ **do**
	$S \leftarrow p^b(S)$
	$S \leftarrow (S_{r_2} \oplus y_i) \,\|\, S_{c_2}$
	Squeeze
	$S \leftarrow p^c(S)$
	$K_A^* \leftarrow \lceil S \rceil^k$
	return K_A^*

Algorithm 4: Encryption and decryption functions.

$\textsc{IsapEnc}_{b,c}^{r_2,r_3}$-$k(K_E, N, P)$	$\textsc{IsapDec}_{b,c}^{r_2,r_3}$-$k(K_E, N, C)$
Input: key $K_E \in \{0,1\}^k$, nonce $N \in \{0,1\}^k$, plaintext $P \in \{0,1\}^*$	**Input:** key $K_E \in \{0,1\}^k$, nonce $N \in \{0,1\}^k$, ciphertext $C \in \{0,1\}^*$
Output: ciphertext $C \in \{0,1\}^*$	**Output:** plaintext $P \in \{0,1\}^*$
$\ell = \lvert N \rvert \mod r_2$	$\ell = \lvert N \rvert \mod r_2$
if $\ell = 0$ **then**	**if** $\ell = 0$ **then**
$N_1 \dots N_u \leftarrow r_2$-bit blocks of N	$N_1 \dots N_u \leftarrow r_2$-bit blocks of N
else	**else**
$N_1 \dots N_u \leftarrow r_2$-bit blocks of $N \,\|\, 0^{r_2-\ell}$	$N_1 \dots N_u \leftarrow r_2$-bit blocks of $N \,\|\, 0^{r_2-\ell}$
$\ell = \lvert P \rvert \mod r_3$	$\ell = \lvert C \rvert \mod r_3$
if $\ell = 0$ **then**	**if** $\ell = 0$ **then**
$P_1 \dots P_v \leftarrow r_3$-bit blocks of P	$C_1 \dots C_v \leftarrow r_3$-bit blocks of C
else	**else**
$P_1 \dots P_v \leftarrow r_3$-bit blocks of $P \,\|\, 0^{r_3-\ell}$	$C_1 \dots C_v \leftarrow r_3$-bit blocks of $C \,\|\, 0^{r_3-\ell}$
$S \leftarrow K_E \,\|\, IV_3$	$S \leftarrow K_E \,\|\, IV_3$
Absorb	**Absorb**
$S \leftarrow p^c(S)$	$S \leftarrow p^c(S)$
$S \leftarrow (S_{r_2} \oplus N_1) \,\|\, S_{c_2}$	$S \leftarrow (S_{r_2} \oplus N_1) \,\|\, S_{c_2}$
for $i = 2, \dots, u$ **do**	**for** $i = 2, \dots, u$ **do**
$S \leftarrow p^b(S)$	$S \leftarrow p^b(S)$
$S \leftarrow (S_{r_2} \oplus N_i) \,\|\, S_{c_2}$	$S \leftarrow (S_{r_2} \oplus N_i) \,\|\, S_{c_2}$
Squeeze	**Squeeze**
for $i = 1, \dots, v$ **do**	**for** $i = 1, \dots, v$ **do**
$S \leftarrow p^c(S)$	$S \leftarrow p^c(S)$
$C_i \leftarrow S_{r_3} \oplus P_i$	$P_i \leftarrow S_{r_3} \oplus C_i$
if $\ell > 0$ **then** $C_v \leftarrow \lceil C_v \rceil^\ell$	**if** $\ell > 0$ **then** $P_v \leftarrow \lceil P_v \rceil^\ell$
return $C_1 \,\|\, \dots \,\|\, C_v$	**return** $P_1 \,\|\, \dots \,\|\, P_v$