ISAP+: ISAP with Fast Authentication

Arghya Bhattacharjee[1], Avik Chakraborti[2], Nilanjan Datta[2], Cuauhtemoc Mancillas-López[3], Mridul Nandi[1]

[1]Indian Statistical Institute, Kolkata, India
[2]TCG Centres for Research and Education in Science and Technology, Kolkata, India
[3] Computer Science Department, CINVESTAV-IPN, Mexico
`bhattacharjeearghya29@gmail.com, avikchkrbrti@gmail.com, nilanjan.datta@tcgcrest.org, cuauhtemoc.mancillas@cinvestav.mx, mridul.nandi@gmail.com`

Abstract. This paper analyses the lightweight, sponge-based NAEAD mode ISAP, one of the finalists of the NIST Lightweight Cryptography (LWC) standardisation project, that achieves high-throughput with inherent protection against differential power analysis (DPA). We observe that ISAP requires 256-bit capacity in the authentication module to satisfy the NIST LWC security criteria. In this paper, we study the analysis carefully and observe that this is primarily due to the collision in the associated data part of the hash function which can be used in the forgery of the mode. However, the same is not applicable to the ciphertext part of the hash function because a collision in the ciphertext part does not always lead to a forgery. In this context, we define a new security notion, named 2PI+ security, which is a strictly stronger notion than the collision security, and show that the security of a class of encrypt-then-hash based MAC type of authenticated encryptions, that includes ISAP, reduces to the 2PI+ security of the underlying hash function used in the authentication module. Next we investigate and observe that a feed-forward variant of the generic sponge hash achieves better 2PI+ security as compared to the generic sponge hash. We use this fact to present a close variant of ISAP, named ISAP+, which is structurally similar to ISAP, except that it uses the feed-forward variant of the generic sponge hash in the authentication module. This improves the overall security of the mode, and hence we can set the capacity of the ciphertext part to 192 bits (to achieve a higher throughput) and yet satisfy the NIST LWC security criteria.

Keywords: Authenticated Encryption, ISAP, ISAP+, Re-keying, Side Channel Resistant, 2PI+, Sponge

1 Introduction

The emergence of side-channel and fault attacks [11, 12, 28, 29] has made it clear that cryptographic implementations may not always behave like a black box. Instead, they might behave like a grey box where the attacker has physical access to the device executing a cryptographic task. As a result, designers have started to design side-channel countermeasures such as masking [13,24]. However, cryptographic primitives like block ciphers (for example, AES [17] and ARX-based designs [7]) are costly to be mask-protected against side-channel attacks. Consequently, designing primitives or modes with inherent side-channel protection is becoming an essential and popular design goal. In this line of design, several block ciphers (e.g., Noekeon [27], PICARO [34], Zorro [22]) and permutations (e.g., ASCON-p [21], KECCAK-p [2,26]) have been proposed with dedicated structures to reduce the resource requirements for masking. In addition, a few NAEAD (Nonce based Authenticated Encryption with Associated Data) modes, such as ASCON [14,21], PRIMATES [3], SCREAM [25], KETJE/KEYAK [10] have been proposed and submitted to the CAESAR [16] competition with the same goal in mind. However, they still lead to significant overheads.

In [31], Medwed et al. have proposed a new technique called fresh *re-keying* to have inherent side-channel protection. Following their work, a series of works [5,5,30] have been proposed that use this novel concept. This technique requires a side-channel resistant fresh key computation function with the nonce and the master key as the inputs. The main idea behind these designs is to ensure that different session keys are used for different nonces. Hence, a new nonce should be used to generate a fresh session key.

1.1 ISAP and Its Variants

In [20], Dobraunig et al. proposed a new authenticated encryption, dubbed ISAP v1, following the re-keying strategy. It is a sponge-based design [8,9] that follows the Encrypt-then-MAC paradigm [6]. We'll traditionally use the terms *rate* and *capacity* to represent the exposed part and the hidden part of the state of the sponge construction respectively. ISAP v1 claims to offer higher-order differential power analysis (DPA) protection provided by an inherent design strategy that combines a sponge-based stream cipher for the encryption module with a sponge-based MAC (suffix keyed) for the authentication module. Both the modules compute fresh session keys using a GGM [23] tree-like function to strengthen the key computation against side-channel attacks.

Later, ISAP v1 was improved to ISAP v2 [15, 19], and was submitted to the NIST Lightweight Cryptography (LWC) standardization project [32] and currently one of the finalists. ISAP v2 is equipped with several promising features and currently is considered to be a strong candidate for the competition. It recommends two variants, instantiated with the lightweight permutations ASCON-p and KECCAK-p[400]. Precisely, ISAP v2 retains all the inherent DPA resistance properties of ISAP v1 along with a better resistance against other implementation based attacks. In addition, ISAP v2 is even more efficient in hardware resources than the first version. ISAP v2 has been highly praised by the cryptography community, and to the best of our knowledge, it is the only inherently DPA protected NAEAD mode that aims to be implemented on lightweight platforms. The ISAP mode is flexible and can be instantiated with any sufficiently large permutation. Precisely, the security claims made by the designers depict that

ISAP needs around 256-bit capacity to satisfy the NIST security criteria and hence needs a permutation with the state size larger than 256-bits. Thus it is highly desirable to analyze the mode further to understand whether it can be designed with a smaller capacity and hence a higher rate, that directly impacts the throughput.

1.2 Improving the Throughput of ISAP

ISAP v2 proposes four instances with the ASCON-p and the KECCAK-p permutations. ISAP v2 with ASCON-p (a 320-bit permutation) is designed with a 64 bit rate. However, it is better to achieve a higher rate design for a higher throughput. The observation is similar for the other instances as well. A potential direction can be to design an algorithm with an improved security bound over the capacity to increase the rate without compromise in the security level. An increased security bound can also help the designers to achieve the same security with a lower state size. This in turn may reduce the register size by using a permutation with a smaller state. A potential choice can be to analyze ISAP with a focus on the BBB (Beyond Birthday Bound) NAEAD security.

We observe that ISAP adopts an efficient approach of applying an unkeyed hash function on the nonce, the associated data, and the ciphertext, and then uses a PRF (Pseudo Random Function) on the hash value. In this case, the security of the NAEAD mode boils down to the collision security of the underlying hash function. This mode can bypass the requirement of storing the master key and can get rid of the key register. However, the hash collision results in a relatively low security bound that may not always be acceptable in ultra-lightweight applications as low security bound forces the designs to adopt primitives with high state size. Thus, an increase in the security bound has the full potential to increase the hardware performance of the design significantly. Motivated by this issue, we aim to study the tightness of the security bound to optimize the throughput and the hardware footprint. Note that, ISAP is already an efficient construction and has been reviewed rigorously by various research groups. Hence, a more detailed mode analysis and any possible mode optimization can further strengthen the construction.

The security proof in [20] by Dobraunig et al. showed that ISAP achieves security up to the birthday bound on the capacity, i.e. of $O(T^2/2^c)$, where T is the time complexity, and c is the capacity size (in bits). We observe that this factor arises due to the simple sponge-type hash applied on the nonce, the associated data, and the random ciphertext. It is obvious to get a collision in the nonce and the associated data that can be trivially used by an adversary to mount a forgery. However, it is not evident how a collision in the random ciphertexts can lead to such an attack. In this regard we investigate the amount of the ciphertext-bit that can be injected per permutation call during the hash and ask the question:

"Can we increase the rate of absorption of the ciphertext blocks in the hash?"

We believe that a positive answer to this question will not only result in a more efficient construction, but more importantly contribute significantly in the direction of NAEAD mode analysis.

1.3 Our Contributions

In this paper, we study a simple variant of ISAP that achieves higher throughput keeping all the primary features of ISAP intact. Our contribution is four-fold:

1. First, in Section 3, we propose a permutation-based generic EtHM (Encrypt then Hash based MAC) type NAEAD mode using a PRF and an unkeyed hash function. This is essentially a generalisation of ISAP type constructions. Note that this generic mode does not guarantee any side-channel resistance; only proper instantiation of the PRF ensures that. In Section 3.2, we have shown that the NAEAD security of EtHM can be expressed in terms of the PRF security of a fixed input length, variable output length keyed function F, the 2PI+ security of H. Intuitively, the 2PI+ notion demands that given a challenge random message of some length chosen by the adversary, it is difficult for an adversary to compute the second pre-image of the random message. We have introduced the notion in Section 2.6. This is in contrast with the traditional collision security that was used for the analysis of ISAP.
2. Next, in Section 4, we first show that for generic sponge hash, a collision attack can be extended to a 2PI+ attack. Thus, the generic sponge hash achieves 2PI+ security of $\Omega(T^2/2^c)$, where c is the capacity of the sponge hash. Next, we consider a feed-forward variant of the sponge hash that uses (i) generic sponge hash to process the nonce and the associated data and (ii) a feed-forward variant of the sponge hash to process the message. We show that the feed-forward property ensures that a collision attack can not be extended to mount a 2PI+ attack. In fact, we prove that this variant of sponge hash obtains an improved security of $O(DT/2^c)$. Note that D and T are data and time complexity respectively, and we typically allow $T \approx D^2$. Hence, feed-forward based sponge achieves a better 2PI+ security as we consider security in terms of D and T instead of traditional one parameter security.
3. Next, in Section 5, we have considered a simple variant of ISAP with minimal changes, named ISAP+, which is a particular instantiation of the generic EtHM mode. To be specific, the differences between ISAP+ and ISAP are as follows:
   (a) Instead of using the generic sponge hash as used in ISAP, we use the feed-forward variant of sponge hash as discussed above.
   (b) In the authentication module of ISAP+, we use the capacity of c' bits for nonce, associated data and first block of ciphertext processing. For rest of the ciphertext blocks, we use capacity of c bits.
   (c) We make a separation among the messages depending on whether its length is less than r' bits or not. The domain separation is performed by adding 0 or 1 to the capacity part before the final permutation call.

This modification ensures that ISAP+ achieves improved security of $O(T^2/2^{c'} + DT/2^c)$, where n is the state-size or the size of the permutation (in bits), $c' = n - r$, $c' = n - r'$). This security boost allows the designer to choose c' and c ($< c'$) effectively to obtain better throughput.

4. Our primary proposal is denoted as ISAP+-A-128, which we have instantiated with ASCON-p. For a fair comparison with ISAP, we have implemented three more variants of ISAP+ in Section 6, comparable with the corresponding ISAP instances. We've instantiated these variants with ASCON-p and KECCAK-p[400] permutations. These variants are denoted as ISAP+-A-128, ISAP+-A-128A, ISAP+-K-128 and ISAP+-K-128A. The first two use full and round reduced variants of ASCON-p and the other two use full and round reduced variants of KECCAK-p[400] respectively. Note that all these instances follow the ISAP+ mode, and they only differ in the choice of the underlying permutation. Also note that these four ISAP+ instances use ASCON-p or KECCAK-p[400] with the same number of rounds as their ISAP counterparts for a fair comparison. Finally, we provide a detail hardware implementation in Section 6 for all the ISAP+ and ISAP instances.

   Note that we propose only one instance of ISAP+ to stick to the security assumption that we use one single permutation in our construction and only the ISAP+-A-128 variant achieves the same. All the other three variants use different number of rounds at different stages of our construction. We have implemented them for a fair comparison with ISAP and to showcase the efficiency of the ISAP+ mode in throughput.

1.4 Relevance of the Work

To understand the relevance of the improved security, let us consider the instantiation of ISAP with ASCON and KECCAK and compare them with ISAP+. To satisfy the NIST requirements, ISAP+ can use $c = 192$, $c' = 256$, and hence, it has a injection rate of $r = 128$-bits for the ciphertexts and an injection rate of $r = 64$ bits for associated data. Table 2 demonstrate a comparative study of ISAP and ISAP+ in terms of the number of permutation calls required for the authentication module.

Table 1: Comparative Study of ISAP+ and ISAP on the no. of permutation calls in the authentication module for associated data of length a bits, message of length m bits.

Mode	Permutation	Parameters	# permutation calls
ISAP	ASCON	$r = 64$	$\lceil \frac{a+m+1}{64} \rceil$
ISAP+	ASCON	$r = 128,\ r' = 64$	$\lceil \frac{a+1}{64} \rceil + \lceil \frac{m}{128} \rceil$
ISAP	KECCAK	$r = 144$	$\lceil \frac{a+m+1}{144} \rceil$
ISAP+	KECCAK	$r = 208,\ r' = 144$	$\lceil \frac{a+1}{144} \rceil + \lceil \frac{m}{208} \rceil$

This result demonstrates that for applications that require long message processing, ISAP+ performs better than ISAP in terms of throughput and speed. Let us consider a concrete example. Consider encrypting a message of length 1 MB with an associate data of length 1 KB using ASCON permutation. With ISAP, the authentication module requires around $1, 31, 201$ many primitive calls. On the other hand, with ISAP+ this requires only $65, 665$ many primitive calls, which is almost half as compared to ISAP.

This paper depicts the robustness of the mode ISAP, and how one can increase the throughtput of the mode at the cost of some hardware area preserving all the inherent security features. This result seems relevant to the cryptography community in the sense that ISAP is also a finalist of the NIST LWC project for standardization.

1.5 Interpretation of Hardware Implementation Result

Both ISAP+ and ISAP have been implemented using VHDL and mapped on Virtex 7XC7V585T (Vivado 2020.2), and the results are summarized in Table 2. The result depicts that all the ISAP+ instances are better in throughput and throughput/area than the ISAP instances with a small compromise in the hardware area. Blue coloured entry is our primary recommendation.

Table 2: FPGA Results of ISAP+ and ISAP

Instances	**Slice Registers**	**LUTs**	**Slices**	**Frequency (MHZ)**	**Encryption Throughput (Gbps)**	**Authentication Throughput (Gbps)**
ISAP-A-128	818	1550	451	400	2.13	2.13
ISAP+-A-128	1081	1742	507	384.16	2.05	3.07
ISAP-K-128	1143	1932	582	300	3.60	2.16
ISAP+-K-128	1423	2245	613	300	3.60	2.64
ISAP-A-128A	810	1539	449	400	4.27	2.13
ISAP+-A-128A	1070	1728	501	384.16	4.09	3.07
ISAP-K-128A	1131	1850	574	300	5.40	2.70
ISAP+-K-128A	1412	2231	605	300	5.40	4.40

2 Preliminaries

2.1 Notations

We'll usually use lowercase letters (e.g., x, y) for integers and indices, uppercase letters (e.g., X, Y) for binary strings and functions, and calligraphic uppercase letters (e.g., $\mathcal{X}$, $\mathcal{Y}$) for sets and spaces. $\mathbb{N}$ and $\mathbb{Z}$ will be used to denote the set of natural numbers and the set of integers respectively. 0^x and 1^y will denote the sequence of x 0's and y 1's respectively. $\{0,1\}^x$ and $\{0,1\}^*$ will denote the set of binary strings of length x and the set of all binary strings respectively. For any $X \in \{0,1\}^*$, $|X|$ and $\|X\|$ will denote the number of bits, and the number of blocks of the binary string X respectively, where the size of the blocks should be clear from the context. For two binary strings X and Y, $X\|Y$ will denote the concatenation of X and Y. For any $X \in \{0,1\}^*$, we define the parsing of X into r-bit blocks as $X_1 \cdots X_x \xleftarrow{}_r X$, where $|X_i| = r$ for all $i < x$ and $1 \leq |X_x| \leq r$ such that $X = X_1\| \cdots \|X_x$. For any $X \in \{0,1\}^*$, $X_1 \cdots X_x \twoheadleftarrow_r X$ does the work of $X_1 \cdots X_x \xleftarrow{}_r X$, and follows it by the compulsory 10^* padding. Given any sequence $X = X_1 \cdots X_x$ and $1 \leq a \leq b \leq x$, we'll represent the subsequence $X_a \ldots X_b$ by $X[a \cdots b]$. For integers $a \leq b$, we'll write $[a \cdots b]$ for the set $\{a, \ldots, b\}$, and for integers $1 \leq a$, we'll write $[a]$ for the set $\{1, \ldots, a\}$. We'll use the notations $\lceil x \rceil_r$ and $\lfloor x \rfloor_r$ to denote the decimal ceiling and floor function on the integer x respectively, and similarly, $\lceil X \rceil_r$ and $\lfloor X \rfloor_r$, to denote the most significant x bits and the least significant x bits of the binary string X respectively. By $X \xleftarrow{\$} \mathcal{X}$, we'll denote that X is chosen uniformly at random from the set $\mathcal{X}$.

2.2 Distinguishing Advantage

For two oracles $\mathcal{O}_0$ and $\mathcal{O}_1$, an algorithm $\mathcal{A}$ which tries to distinguish between $\mathcal{O}_0$ and $\mathcal{O}_1$ is called a distinguishing adversary. $\mathcal{A}$ plays an interactive game with $\mathcal{O}_b$ where b is unknown to $\mathcal{A}$, and then outputs a bit $b_{\mathcal{A}}$. The winning event is $[b_{\mathcal{A}} = b]$. The distinguishing advantage of $\mathcal{A}$ is defined as

$$\mathbf{Adv}_{\mathcal{O}_1,\mathcal{O}_0}(\mathcal{A}) := \left| \Pr[b_{\mathcal{A}} = 1 | b = 1] - \Pr[b_{\mathcal{A}} = 1 | b = 0] \right| .$$

Let $\mathbf{A}[q,t]$ be the class of all distinguishing adversaries limited to q oracle queries and t computations. We define

$$\mathbf{Adv}_{\mathcal{O}_1,\mathcal{O}_0}[q,t] := \max_{\mathbf{A}[q,t]} \mathbf{Adv}_{\mathcal{O}_1,\mathcal{O}_0}(\mathcal{A}) .$$

When the adversaries in $\mathbf{A}[q,t]$ are allowed to make both encryption queries and decryption queries to the oracle, this is written as $\mathbf{Adv}_{\pm\mathcal{O}_1,\pm\mathcal{O}_0}[q,q',t]$, where q is the maximum number of encryption queries allowed and q' is the maximum number of decryption queries allowed. Enc_b and Dec_b denote the encryption and the decryption function associated with $\mathcal{O}_b$ respectively. $\mathcal{O}_0$ conventionally represents an ideal primitive, while $\mathcal{O}_1$ represents either an actual construction or a mode of operation built of some other ideal primitives. Typically the goal of the function represented by $\mathcal{O}_1$ is to emulate the ideal primitive represented by $\mathcal{O}_0$. We use the standard terms real oracle and ideal oracle for $\mathcal{O}_1$ and $\mathcal{O}_0$ respectively. A security game is a distinguishing game with an optional set of additional restrictions, chosen to reflect the desired security goal. When we talk of distinguishing advantage with a specific security game $\mathcal{G}$ in mind, we include $\mathcal{G}$ in the superscript, e.g., $\mathbf{Adv}^{\mathcal{G}}_{\mathcal{O}_1,\mathcal{O}_0}(\mathcal{A})$. Also we sometimes drop the ideal oracle and simply write $\mathbf{Adv}^{\mathcal{G}}_{\mathcal{O}_1}(\mathcal{A})$ when the ideal oracle is clear from the context.

2.3 Authenticated Encryption and Its Security Notion

A **N**once based **A**uthenticated **E**ncryption with **A**ssociated **D**ata (**NAEAD**) involves a key space $\mathcal{K}$, a nonce space $\mathcal{N}$, an associated data space $\mathcal{AD}$, a message space $\mathcal{M}$ and a tag space $\mathcal{T}$ along with two functions $\mathsf{Enc} : \mathcal{K} \times \mathcal{N} \times \mathcal{AD} \times \mathcal{M} \rightarrow \mathcal{M} \times \mathcal{T}$ (called the Encryption Function) and $\mathsf{Dec} : \mathcal{K} \times \mathcal{N} \times \mathcal{AD} \times \mathcal{M} \times \mathcal{T} \rightarrow \mathcal{M} \cup \{\perp\}$ (called the Decryption Function) with the correctness condition that for any $K \in \mathcal{K}, N \in \mathcal{N}, A \in \mathcal{AD}$ and $M \in \mathcal{M}$, it holds that

$$\mathsf{Dec}(K, N, A, \mathsf{Enc}(K, N, A, M)) = M\,.$$

In the NAEAD security game, the real oracle involves such a pair of functions Enc_1 and Dec_1 with $K \xleftarrow{\$} \mathcal{K}$. On the other hand, the ideal oracle involves an ideal random function $\mathsf{Enc}_0 : \mathcal{K} \times \mathcal{N} \times \mathcal{AD} \times \mathcal{M} \rightarrow \mathcal{M} \times \mathcal{T}$ and a constant function $\mathsf{Dec}_0 : \mathcal{K} \times \mathcal{N} \times \mathcal{AD} \times \mathcal{M} \times \mathcal{T} \rightarrow \{\perp\}$. The adversary ($\mathcal{A}$) which interacts with one of the two oracles is supposed to be:

1. Nonce-respecting, i.e., $\mathcal{A}$ should not repeat a nonce in more than one encryption queries, and
2. Non-repeating, i.e., $\mathcal{A}$ should not make the decryption query (N, A, C, T) if it has already made the encryption query (N, A, M) and received (C, T) in response.

The distinguishing advantage of $\mathcal{A}$ will be denoted by $\mathbf{Adv}^{\mathrm{NAEAD}}_{(\mathsf{Enc}_1,\mathsf{Dec}_1)}(\mathcal{A})$. The following two security notions are captured in this advantage.

1. Privacy or Confidentiality, i.e., $\mathcal{A}$ should not be able to distinguish the real oracle from the ideal oracle.
2. Authenticity or Integrity, i.e., $\mathcal{A}$ should not be able to forge the real oracle. In other words, $\mathcal{A}$ should not be able to make a decryption query to the real oracle to which the response isn't $\perp$.

2.4 The Coefficients H Technique

The Coefficients H Technique is a proof method by Patarin [33]. Consider two oracles $\mathcal{O}_0$ (the ideal oracle) and $\mathcal{O}_1$ (the real oracle). Let $\mathcal{T}$ denote the set of all possible transcripts (i.e., the set of all query-response pairs) an adversary can obtain. For any transcript $\tau \in \mathcal{T}$, we will denote the probability to realize the transcript as $\mathsf{ip}_{\mathsf{real}}(\tau)$ or $\mathsf{ip}_{\mathsf{ideal}}(\tau)$ when it is interacting with the real or the ideal oracle respectively. We call them the interpolation probabilities. W.l.o.g., we assume that the adversary is deterministic. Hence, the interpolation probabilities are the properties of the oracles only. As we deal with stateless oracles, these probabilities are independent of the order of the query-response pairs in the transcript.

Theorem 1. *Suppose for a set $\mathcal{T}_{good} \subseteq \mathcal{T}$ of transcripts (called the* **good transcripts***) the following holds:*

1. *For any adversary $\mathcal{A}$ interacting with $\mathcal{O}_0$ (the ideal oracle), the probability of getting a transcript in $\mathcal{T}_{good}$ is at least $1 - \epsilon_{\mathsf{bad}}$. We may denote the set $\mathcal{T} \setminus \mathcal{T}_{good}$ by $\mathcal{T}_{bad}$. Hence, the probability of getting a transcript in $\mathcal{T}_{bad}$ is at most ϵ_{bad}.*

2. For any adversary $\mathcal{A}$ and for any transcript $\tau \in \mathcal{T}_{good}$,

$$\mathsf{ip}_{\mathsf{real}}(\tau) \geq (1 - \epsilon_{\mathit{ratio}}) \cdot \mathsf{ip}_{\mathsf{ideal}}(\tau).$$

For an oracle $\mathcal{O}_1$ (the real oracle) satisfying (1) and (2), we have

$$\mathbf{Adv}_{\mathcal{O}_0,\mathcal{O}_1}(\mathcal{A}) \leq \epsilon_{\mathit{bad}} + \epsilon_{\mathit{ratio}}.$$

2.5 Fixed Input - Variable Output PRFs with Prefix Property

A fixed input variable output function (FIL-VOL) is a keyed function $F_K : \{0,1\}^\star \times \{0,1\} \times \mathbb{N} \rightarrow \{0,1\}^\star$ that takes as input an input a string $I \in \{0,1\}^\star$, a flag $b \in \{0,1\}$ as input, a positive integer $\ell \in \mathbb{N}$, and outputs a string $O \in \{0,1\}^\ell$, i.e., $O := F_K(I, b; \ell)$. We call such a keyed function a FIL-VOL pseudo random function maintaining the prefix-property if

- for all inputs $(I, b; \ell), (I', b'; \ell')$ with $(I, b) \neq (I', b')$, $F_K(I, b; \ell)$, $F_K(I', b'; \ell)$ are distributed uniformly at random, and
- for all inputs $(I, b; \ell), (I, b; \ell')$, with $\ell' > \ell$, $\lceil F_K(I', b'; \ell') \rceil_\ell = F_K(I', b'; \ell)$.

More formally,

$$\mathbf{Adv}_F^{\mathsf{PRF}}(\mathcal{A}) := |\Pr[\mathcal{A}^{F_K} = 1] - \Pr[\mathcal{A}^f = 1]|,$$

where f is a random function from same domain and range maintaining the prefix property.

2.6 Multi-Target 2nd Pre-Image with Associated Data

In this section, we discuss the notion of multi-target 2nd pre-image security of permutation-based hash functions.

In this setting, an adversary (say $\mathcal{A}$) chooses q (nonce, associated data, length)-tuples to the challenger $\mathcal{C}$, say $(N_i, A_i, \ell_i)_{i=1..q}$. The challenger in turn returns q uniformly random messages of specified lengths respectively, say $C_1, \ldots, C_q$. The queries $(N_i, A_i, C_i)_{i=1..q}$ are called *challenge queries.* The goal of $\mathcal{A}$ is to return q' many $(N'_j, A'_j, C'_j)_{j=1..q'}$ (called response queries) tuples such that at least one of the hash values of (N'_j, A'_j, C'_j) matches with the hash value of any one of the (N_i, A_i, C_i). Note that the adversaries are allowed to set some challenge queries as response queries: $(N'_j, A'_j, C'_j) = (N_i, A_i, C_i)$, for some i, j. However, for the winning event the challenge and response queries should be distinct. The adversary can make up to q_p queries to p or p^{-1}. Formally, the advantage of $\mathcal{A}$ is defined as

$$\begin{aligned}\mathbf{Adv}_{\mathrm{H}}^{\mathsf{2PI+}}(\mathcal{A}) := \Pr[\exists i, j,\ & H^p(N'_j, A'_j, C'_j) = H^p(N_i, A_i, C_i), \\ & (N'_j, A'_j, C'_j) \neq (N_i, A_i, C_i)].\end{aligned}$$

where H is an IV-based hash function. Note that the adversary is allowed to make hash queries before, after, or in between its interaction with the challenger to obtain the challenge message(s). Also, note that the 2PI+ security does not depend on the message length. The fact that the adversary submits a length ℓ_i to the challenger to obtain each message before the submission of the challenge message is merely because the 2PI+ security notion enables its adversary to obtain messages of whatever lengths it pleases.

3 An EtHM Paradigm for NAEAD

This section introduces an efficient generalized Encrypt-then-Hash based MAC (EtHM) paradigm for NAEAD modes. This is a generalized paradigm for constructing side-channel resilient modes such as ISAP.

3.1 Specification

Let n, k and τ be positive integers such that $n > \tau$. The construction takes as input a plaintext M, a nonce N, an associated data A, and outputs a ciphertext C and a tag T. Given a permutation based FIL-VOL keyed-function with prefix property F_K^p, and a permutation based un-keyed hash function $H^p : \{0,1\}^* \to \{0,1\}^n$ the mode works as follows.

$$\begin{aligned} C &= M \oplus F_K^p(N, 0; |M|), \\ T\|D &= p(F_K^p(X, 1; |X|)\|Z), \text{ where } X\|Z = H^p(N, A, C)\,. \end{aligned}$$

The authenticated encryption module is pictorially depicted in Figure 1. Note that T denotes the most significant τ bits of the output of the permutation call. The least significant $(n - \tau)$ bits is denoted by D. Note that we do not need D from the construction point of view, however, we require it during the security analysis. Notations F, H and F_K^p, H^p have been used in this paper interchangeably for convenience, and aren't supposed to create any confusion. From time to time, we'll address this paradigm as EtHM only.

3.2 Security of EtHM

In this subsection, we analyse the NAEAD security of EtHM with F as the underlying function and H as the multi-target IV-respecting second pre-image resistant hash function. Formally, we prove the following theorem.

Theorem 2 (NAEAD Security of EtHM). *Consider EtHM based on a function F and a hash function H. For all deterministic nonce-respecting non-repeating query making adversary $\mathcal{A}$ which can make at most q_e encryption queries, q_v decryption queries and q_p primitive queries to p and its inverse and assuming $q = q_e + q_v$, there exists two adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ such that the NAEAD advantage of $\mathcal{A}$ can be bounded by*

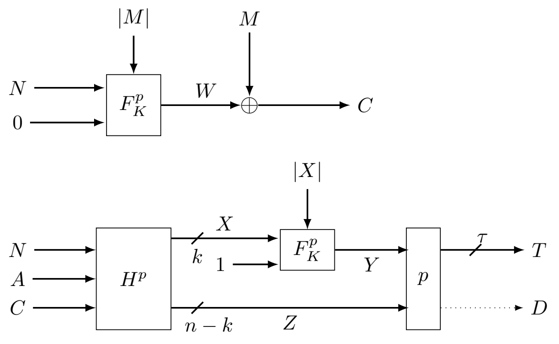

Fig. 1: Authenticated Encryption module of the EtHM paradigm.

$$\mathbf{Adv}_{\mathit{EtHM}}^{NAEAD}(\mathcal{A}) \leq \mathbf{Adv}_{F}^{\mathit{PRF}}(\mathcal{B}_1) + 2\mathbf{Adv}_{\lfloor H \rfloor_{n-k}}^{\mathit{2PI+}}(\mathcal{B}_2) + \frac{qq_p}{2^n} + \frac{2kq_v}{2^k} + \frac{q_v}{2^\tau}$$
$$+ \frac{\binom{q_p}{k}}{2^{\tau(k-1)}} + \frac{\binom{q_p}{k}}{2^{(n-k)(k-1)}},$$

where $\mathcal{B}_1$ can make $2q$ PRF queries and $\mathcal{B}_2$ can make q challenge queries, q response queries and q_p primitive queries to p and its inverse.

Proof. Let $\mathbf{Enc}^{F_K^p,p}$ and $\mathbf{Dec}^{F_K^p,p}$ be the encryption and the decryption function of EtHM respectively. Let us call its oracle $\mathcal{O}_1 = (\mathbf{Enc}^{F_K^p,p}, \mathbf{Dec}^{F_K^p,p}, p)$. We have to upper-bound the distinguishing advantage of $\mathcal{A}$ interacting with $\mathcal{O}_1$ or $\mathcal{O}_3 = (\$, \perp, p)$. For our purpose, we define an intermediate oracle by replacing F_K^p in $\mathcal{O}_1$ by a random functions \$. Let us call this new intermediate oracle $\mathcal{O}_2 = (\mathbf{Enc}^{\$,p}, \mathbf{Dec}^{\$,p}, p)$. We will employ a standard reduction proof. We break down the distinguishing game of $\mathcal{A}$ using the triangle inequality as follows.

$$\begin{aligned}\mathbf{Adv}_{\mathsf{EtHM}}^{\mathrm{NAEAD}}(\mathcal{A}) &= |\Pr[\mathcal{A}^{\mathcal{O}_1} = 1] - \Pr[\mathcal{A}^{\mathcal{O}_3} = 1]| \\ &\leq |\Pr[\mathcal{A}^{\mathcal{O}_1} = 1] - \Pr[\mathcal{A}^{\mathcal{O}_2} = 1]| \\ &+ |\Pr[\mathcal{A}^{\mathcal{O}_2} = 1] - \Pr[\mathcal{A}^{\mathcal{O}_3} = 1]| \,. \qquad (1)\end{aligned}$$

Now, we bound each of the two terms.
□ Bounding $|\Pr[\mathcal{A}^{\mathcal{O}_1} = 1] - \Pr[\mathcal{A}^{\mathcal{O}_2} = 1]|$. We bound this term by the PRF advantage of F. For that, let us consider the following adversary $\mathcal{B}_1$ that runs $\mathcal{A}$ (any distinguisher of $\mathcal{O}_1$ and $\mathcal{O}_2$) as follows.

– Whenever $\mathcal{A}$ submits an encryption query (N, A, M), $\mathcal{B}_1$ submits $(N, |M|, 0)$ to its challenger. Suppose the challenger returns C. $\mathcal{B}_1$ calculates $X\|Z = H^p(N, A, C)$ with $|X| = k$ and $|Z| = n - k$ and submits $(X, 1; k)$ to its challenger. Suppose the challenger returns Y. $\mathcal{B}_1$ returns $(C, p(Y\|Z))$ to $\mathcal{A}$.
– Similarly, whenever $\mathcal{A}$ submits a decryption query $(\hat{N}, \hat{A}, \hat{C}, \hat{T})$, $\mathcal{B}_1$ submits $(\hat{N}, 0; |\hat{C}|)$ to its challenger. Suppose the challenger returns $\hat{M}$. $\mathcal{B}_1$ calculates $\hat{X}\|\hat{Z} = H^p(\hat{N}, \hat{A}, \hat{C})$ with $|\hat{X}| = k$ and $|\hat{Z}| = n - k$ and submits $(\hat{X}, 1; k)$ to its challenger. Suppose the challenger returns $\hat{Y}$. $\mathcal{B}_1$ calculates $\lceil p(\hat{Y}\|\hat{Z})\rceil_\tau$ If $T = \hat{T}$, then $\mathcal{B}_1$ returns $\hat{M}$ to $\mathcal{A}$. Otherwise it returns $\perp$.
– At the end of the game, $\mathcal{A}$ submits the decision bit to $\mathcal{B}_1$ which it forwards to its challenger. Note that when $\mathcal{A}$ supposedly interacts with $\mathcal{O}_1$ or $\mathcal{B}_1$ supposedly interacts with F_K^p, they submit $b = 1$. Otherwise, they submit $b = 0$.

It is easy to see $\Pr[\mathcal{A}^{\mathcal{O}_1} = 1] = \Pr[\mathcal{B}_1^{F_K^p} = 1]$ and $\Pr[\mathcal{A}^{\mathcal{O}_2} = 1] = \Pr[\mathcal{B}_1^{\$} = 1]$, and hence we obtain the following.

$$|\Pr[\mathcal{A}^{\mathcal{O}_1} = 1] - \Pr[\mathcal{A}^{\mathcal{O}_2} = 1]| = \mathbf{Adv}_F^{\mathsf{PRF}}(\mathcal{B}_1). \tag{2}$$

□ Bounding $|\Pr[\mathcal{A}^{\mathcal{O}_2} = 1] - \Pr[\mathcal{A}^{\mathcal{O}_3} = 1]|$. This bound follows from the lemma given below, the proof of which is deferred to the next section.

Lemma 1. *Let $\mathcal{A}$ be a deterministic nonce-respecting non-repeating query making adversary interacting with oracle $\mathcal{O}_2$ or $\mathcal{O}_3$ which can make at most q_e encryption queries, q_v decryption queries and q_p primitive queries to p and its inverse. Assuming $q = q_e + q_v$, there exists an adversary $\mathcal{B}_2$ such that the NAEAD advantage of $\mathcal{A}$ can be bounded by*

$$|\Pr[\mathcal{A}^{\mathcal{O}_2} = 1] - \Pr[\mathcal{A}^{\mathcal{O}_3} = 1]| \leq 2\mathbf{Adv}_{\lfloor H \rfloor_{n-k}}^{\mathit{2PI+}}(\mathcal{B}_2) + \frac{qq_p}{2^n} + \frac{2kq_v}{2^k} + \frac{q_v}{2^\tau} + \frac{\binom{q_p}{k}}{2^{\tau(k-1)}} + \frac{\binom{q_p}{k}}{2^{(n-k)(k-1)}},$$

where $\mathcal{B}_2$ can make q challenge queries, q response queries and q_p primitive queries to p and its inverse.

The proof of the theorem follows from Equation 1, Equation 2 and Lemma 1.

3.3 Proof of Lemma 1

Now we'll prove Lemma 1 using Coefficients H Technique step by step.
Step I: Sampling of the Ideal Oracle and Defining the Bad Events. We start with sampling of the ideal oracle and go on mentioning the bad events whenever they occur. Note that whenever we mention a bad event, even if it's not explicitly mentioned, it's implicitly understood that the previous bad events haven't occurred.

In the online phase, the adversary interacts with the oracles and receives the corresponding responses. In this phase, it can make any construction or permutation query. The i-th encryption query is (N^i, A^i, M^i), the i-th decryption query is $(\hat{N}^i, \hat{A}^i, \hat{C}^i, \hat{T}^i)$, $H(\hat{N}^i, \hat{A}^i, \hat{C}^i) = \hat{X}^i \| \hat{Z}^i$ with $|\hat{X}^i| = k$ and $|\hat{Z}^i| = n-k$, the i-th permutation query is U^i if it's a forward query (i.e., p query), and V^i if it's a backward query (i.e., p^{-1} query).

1. Return (C^i, T^i), $\forall i \in [q_e]$, where $C^i \xleftarrow{\$} \{0,1\}^{|M^i|}, T^i \xleftarrow{\$} \{0,1\}^{\tau}$.
2. Return $\perp, \forall i \in [q_v]$.
3. Return the true output values of the permutation queries.
4. Set $X^i := \lceil H(N^i, A^i, C^i) \rceil_k, \hat{X}^i := \lceil H(\hat{N}^i, \hat{A}^i, \hat{C}^i) \rceil_k$,
   $Z^i := \lfloor H(N^i, A^i, C^i) \rfloor_{n-k}, \hat{Z}^i := \lfloor H(\hat{N}^i, \hat{A}^i, \hat{C}^i) \rfloor_{n-k}$

The adversary aborts if the following (bad) event occurs.

– bad1: $\exists i \in [q_e]$ and $j \in [q_v]$ with $i \neq j$ and $(N^i, A^i, C^i) \neq (\hat{N}^j, \hat{A}^j, \hat{C}^j)$ such that $Z^i = \hat{Z}^j$.
– bad2: $\exists i, j \in [q_e]$ with $i \neq j$ such that $Z^i = Z^j$.

In the offline phase, the adversary can no longer interact with any oracle, but the challenger may release some additional information to the adversary before it submits its decision.

1. $Y^i \xleftarrow{\$} \{0,1\}^k$, $\forall i \in [q_e]$ and $j \in [i-1]$ with $X^i \neq X^j$.
2. $\hat{Y}^i \xleftarrow{\$} \{0,1\}^k$, $\forall i \in [q_v], j \in [i-1]$ and $\ell \in [q_e]$ with $\hat{X}^i \neq \hat{X}^j$ and $\hat{X}^i \neq X^\ell$.

Again, the adversary aborts if any of the following (bad) events occur.

– bad3: $\exists i \in [q_e]$ and $j \in [q_p]$ such that $Y^i \| Z^i = U^j$.
– bad4: There is a k-multi-collision at the τ most significant bits of the output of the forward permutation queries.
– bad5: There is a k-multi-collision at the $(n-k)$ least significant bits of the output of the backward permutation queries.
– bad6: $\exists i \in [q_v]$ and $j \in [q_p]$ such that $\hat{Y}^i \| \hat{Z}^i = U^j$.

If none of the bad events occur, then

1. $D^i \xleftarrow{\$} \{0,1\}^{n-\tau}$, $\forall i \in [q_e]$,
2. $\hat{T}'^i \| \hat{D}^i \xleftarrow{\$} \{0,1\}^n$, $\forall i \in [q_v]$.

Again, the adversary aborts if any of the following (bad) event occurs.

– bad7: $\exists i \in [q_v]$ such that $\hat{T}^i = \hat{T}'^i$.

Step II: Bounding the Probability of the Bad Events. Now we'll upper bound the probabilities of the bad events.

– bad1: This event says that the capacity part of the hash of an encryption query matches with the capacity part of the hash of a forging query. This is nothing but computing a second pre-image corresponding to a challenge (N, A, C), where C is chosen uniformly at random. Thus, the probability of this event is bounded by the 2PI+ security of H.

$$\Pr[\mathsf{bad1}] \leq \mathbf{Adv}^{\mathsf{2PI+}}_{\lfloor H \rfloor_{n-k}}(\mathcal{B}_2)\,,$$

where $\mathcal{B}_2$ can make q challenge queries, q response queries and q_p primitive queries to p and its inverse.

– bad2: This event says that the capacity part of the hash of an encryption query matches with the capacity part of the hash of another encryption query. This is again nothing but computing a second pre-image corresponding to a challenge (N, A, C), where C is chosen uniformly at random. Thus, the probability of this event is bounded by the 2PI+ security of H.

$$\Pr[\mathsf{bad2}] \leq \mathbf{Adv}^{\mathsf{2PI+}}_{\lfloor H \rfloor_{n-k}}(\mathcal{B}_2)\,,$$

where $\mathcal{B}_2$ can make q challenge queries, q response queries and q_p primitive queries to p and its inverse.

– bad3: For a fixed encryption query and a fixed permutation query, the probability of this event comes out to be equal to $1/2^n$ due to the randomness of U^j. Applying union bound over all possible choices, we obtain

$$\Pr[\mathsf{bad3}] \leq \frac{q_e q_p}{2^n}\,.$$

– bad4: For a fixed k-tuple of forward permutation queries, the probability of this event comes out to be equal to $1/2^{\tau(k-1)}$ due to the randomness of the permutation output. Applying union bound over all possible choices, we obtain

$$\Pr[\mathsf{bad4}] \leq \frac{\binom{q_p}{k}}{2^{\tau(k-1)}}\,.$$

– bad5: For a fixed k-tuple of backward permutation queries, the probability of this event comes out to be equal to $1/2^{(n-k)(k-1)}$ due to the randomness of the permutation output. Applying union bound over all possible choices, we obtain

$$\Pr[\mathsf{bad5}] \leq \frac{\binom{q_p}{k}}{2^{(n-k)(k-1)}}\,.$$

– bad6: We analyse this bad event in the three following sub-cases.

  • In this case, the number of multi-collision at the τ most significant bits of the output of the forward permutation queries is at most k. So the adversary can make a hash query (N, A, C) to obtain $X \| Z$, fix Z as the least significant bits and vary the rest of the bits to obtain the multi-collision. Suppose the multi-collision happens at the value T. In that case, if the adversary makes the decryption query (N, A, C, T), then the probability of bad6 comes out to be equal to $k/2^k$. For q_v decryption queries, this probability comes out to be equal to $kq_v/2^k$.

- In this case, the number of multi-collision at the $(n-k)$ least significant bits of the output of the backward permutation queries is at most k. So the adversary can fix the τ most significant bits (say T) and vary the rest of the bits to obtain the multi-collisions. Suppose the multi collisions happen at the values $Z_1, Z_2, \cdots, Z_m$. Also suppose that the adversary has q_1 hash pre-images of Z_1, q_2 hash pre-images of Z_2, $\cdots$, q_m hash pre-images of Z_m, where $q_1+q_2+\cdots+q_m = q_v$. For $i \in [r]$, suppose the adversary has a pre-image (N, A, C) of Z_i. In that case, if the adversary makes the decryption query (N, A, C, T), then the probability of bad6 comes out to be equal to $k/2^k$. For q_v pre-images, this probability comes out to be equal to $kq_v/2^k$.
- If the previous two cases don't occur, i.e., there is no multi-collision, then for a fixed decryption query and a fixed permutation query, he probability of bad6 comes out to be equal to $1/2^n$ due to the randomness of U^j. For q_v decryption queries and q_p permutation queries, this probability comes out to be equal to $q_vq_p/2^n$.

Combining all the three cases, we obtain

$$\Pr[\mathsf{bad6}|(\overline{\mathsf{bad3}} \wedge \overline{\mathsf{bad4}} \wedge \overline{\mathsf{bad5}})] \leq \frac{2kq_v}{2^k} + \frac{q_vq_p}{2^n}\,.$$

– bad7: For a fixed decryption query, the probability of this event comes out to be equal to $1/2^\tau$ due to the randomness of $\hat{T}'^i$. Applying union bound over all possible choices, we obtain

$$\Pr[\mathsf{bad7}] \leq \frac{q_v}{2^\tau}\,.$$

Combining everything, we obtain

$$\begin{aligned}\epsilon_{\mathsf{bad}} := \Pr[\mathsf{bad}] &\leq \Pr[\mathsf{bad1} \vee \mathsf{bad2} \vee \cdots \vee \mathsf{bad7}] \\ &\leq 2\mathbf{Adv}^{\mathsf{2PI+}}_{\lfloor H \rfloor_{n-k}}(\mathcal{B}_2) + \frac{qq_p}{2^n} + \frac{2kq_v}{2^k} + \frac{q_v}{2^\tau} \\ &+ \frac{\binom{q_p}{k}}{2^{\tau(k-1)}} + \frac{\binom{q_p}{k}}{2^{(n-k)(k-1)}}. \end{aligned} \tag{3}$$

Step III: Ratio of Good Interpolation Probabilities. We recall that to obtain oracle $\mathcal{O}_2$, we replace the function F of $\mathcal{O}_1$ with a random function \$. All the remaining specification of $\mathcal{O}_2$ are similar to $\mathcal{O}_1$ (see Section 3.1). Let q_x be the number of construction queries with distinct X^i's and $\hat{X}^i$'s and q' be the number of construction queries with distinct (N^i, A^i, C^i)'s and $(\hat{N}^i, \hat{A}^i, \hat{C}^i)$'s. For any good transcript τ, we get

$$\Pr_{\mathcal{O}_2}[\tau] = \frac{1}{2^{n\sigma_e}} \frac{1}{2^{kq_x}} \frac{1}{(2^n)_{q'+q_p}}\,.$$

The first term corresponds to the number of choices for W^i. The second term corresponds to the number of choices for Y^i. The third term corresponds to the number of choices for the outputs of the distinct permutation calls. We also get

$$\Pr_{\mathcal{O}_3}[\tau] = \frac{1}{2^{n\sigma_e}} \frac{1}{2^{nq'}} \frac{1}{2^{kq_x}} \frac{1}{(2^n)_{q_p}} .$$

The first term corresponds to the number of choices for C^i. The second term corresponds to the number of choices for $T^i \| D^i$. The third term corresponds to the number of choices for Y^i. The fourth term corresponds to the number of choices for the outputs of the distinct permutation calls. Thus we finally obtain

$$\frac{\Pr_{\mathcal{O}_2}[\tau]}{\Pr_{\mathcal{O}_3}[\tau]} \geq 1 \,, \text{i.e., } \epsilon_{\mathsf{good}} = 0. \tag{4}$$

Step IV: Final Calculation. The Lemma follows as we use Equation 3 and Equation 4 in Theorem 1.

4 Multi-Target 2nd Pre-Image Security of Sponge Based Hashes

This section analyses the 2PI+ security of the sponge hash and some of it's variants.

4.1 Sponge Hash and Its 2PI+ Security

First we briefly revisit the sponge hash. Consider the initial state to be $N\|IV$ for some fixed IV. Let $p \in \mathsf{Perm}$ where Perm is the set of all permutations on $\{0,1\}^n$. We call the r most significant bits of the state as rate and the c' least significant bits of the state as capacity. The associated data A and the message C is absorbed in r'-bit blocks by subsequent p-calls, and the output of the last p-call is the hash output T. Figure 2 illustrates the sponge hash. Now let us look at its 2PI+ security.

The following attack demonstrates that the sponge hash is vulnerable to a meet-in-the-middle attack as follows.

- Suppose an adversary (say $\mathcal{A}$) submits $(N, A, 2)$ and receives the random message $C_1\|C_2$ from its challenger where $|C_1| = |C_2| = r'$.
- $\mathcal{A}$ computes the hash as $H = p(p(S_1 \oplus C_1\|S_2 \oplus 0^\star 1) \oplus (C_2\|0^c))$. Suppose $H = p(Y_2\|Z_2)$ where $|Y_2| = r'$ and $|Z_2| = c$.
- $\mathcal{A}$ makes some p-queries of the form $\star\|IV$ and some p^{-1}-queries of the form $\star\|Z_2$, and stores the p-query outputs in the list $\mathcal{L}_1$ and the p^{-1}-query outputs in the list $\mathcal{L}_2$.
- Suppose the capacity of one entry in $\mathcal{L}_1$ (say $Y_1\|Z_1$ where $|Y_1| = r'$ and $|Z_1| = c'$) matches with the capacity of one entry in $\mathcal{L}_2$ (say $Y_1^\star\|Z_1$). Suppose $p(N'\|IV) = Y_1\|Z_1$ and $p^{-1}(Y_2^\star\|Z_2) = Y_1^\star\|Z_1$.

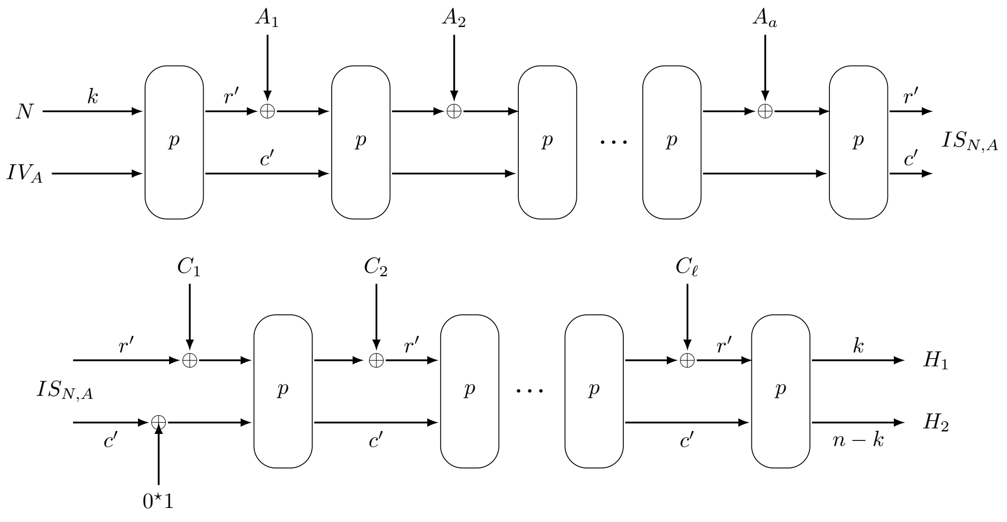

Fig. 2: Sponge Hash with ℓ Message Blocks.

– $\mathcal{A}$ returns $(N', \epsilon, (Y_1 \oplus Y_1^\star)\|(Y_2 \oplus Y_2^\star))$ to its challenger as the second preimage of the random message $(N, C_1\|C_2)$.

It is easy to see that the attack succeeds with probability $\frac{|\mathcal{L}_1\|\mathcal{L}_2|}{2^{c'}}$. In other words, if the adversary is able to make around $2^{c'/2}$ p-queries and p^{-1}-queries each, it would be able to mount this 2PI+ attack with very high probability. Thus, for the sponge hash, the 2PI+ security reduces to the collision security due to the above meet-in-the-middle attack, and the 2PI+ security for sponge hash is $\Omega(q_p^2/2^{c'})$. Now, we are more interested in some other hash functions where a such collision attack doesn't induce a 2PI+ attack.

4.2 Feed Forward Based Sponge Hash and Its 2PI+ Security

Now, we consider a feed forward variant of the sponge hash. The nonce and associated data processing remains as it is. However, the following modifications during the random message processing:

– The capacity part of the output of the i-th permutation is xored with the previous state capacity to obtain the updated i-th state capacity.
– The message injection rate for the first block of random ciphertext remains r' bits, and for all successive blocks the rate is r bits, where $r \geq r'$. To make things compatible, the capacity part before the first p-call is chopped to the least significant c-bits while feed-forwarding.
– We use a domain separation before the final permutation call depending on the size of the random message. If the size is less than or equal to r', we xor 1 in the capacity.

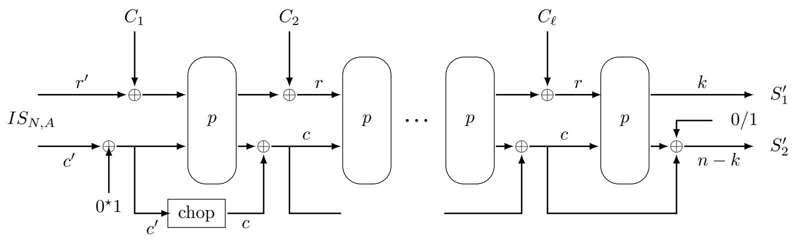

Fig. 3: The Feed Forward Variant of the Sponge Hash. The initial state $IS_{N,A}$ is generated identically as in the sponge hash, depicted in Fig. 2.

Figure 3 illustrates the feed forward variant of the sponge hash with n-bit hash value $H_1\|H_2$. It is easy to see that the attack on the sponge hash can not be extended to this hash due to the feed-forward functionality of this hash. Now let us look at its 2PI+ security. Formally, we state the following lemma:

Lemma 2. *Let H be the feed-forward based sponge hash as defined as above. The 2PI+ security of the construction is given by*

$$\mathbf{Adv}^{\textsf{2PI+}}_{\lfloor H \rfloor_{n-k}}(\mathcal{A}) \le \frac{q_p^2}{2^{c'}} + \frac{(q_p+\sigma_v)\sigma_e}{2^c} + \frac{\sigma_e^2}{2^c} + \frac{q_v}{2^{n-k}},$$

where $\mathcal{A}$ makes at most q_e challenge queries with an aggregate of σ_e blocks, q_v forging attempt queries with an aggregate of σ_v blocks, and q_p many permutation queries.

Proof. First let us consider the scenario for all challenge queries with $\ell \ge r'$. Suppose at the i-th step, the adversary (say $\mathcal{A}$) submits the i-th message length and receives the random message C^i from its challenger. $\mathcal{A}$ makes successive queries to p to derive the hash value corresponding to the fixed IV and C^i. Moreover, the adversary makes several additional queries to p or p^{-1}.

Graph based Representation. Now we draw a graph corresponding to all the challenge, permutation queries and forging attempts made by the adversary $\mathcal{A}$. A node of the graph is an n-bit state value. For a challenge or response query, we consider all the permutation inputs as nodes. Suppose the $(i-1)^{th}$ and i^{th} permutation inputs are X_{i-1}, and X_i respectively, then we draw an edge from node X_{i-1} to X_i with edge labelled as C_i, where C_i is i^{th} message injected. The starting vertex for each query (N, A, C) is defined as $IS_{N,A} \oplus (C_1\|0)$. Now we consider the direct permutation queries. suppose $\mathcal{A}$ makes a p query with the input X, and the output is Y (i.e., $Y = p(X)$), then we draw an edge from vertex X to vertex $Y \oplus (0\|\lfloor X \rfloor_c)$. Similarly, if $\mathcal{A}$ makes a p^{-1} query with input $Y^\star$, and the output is $X^\star$ (i.e., $p^{-1}(Y^\star) = X^\star$), we draw an edge from $X^\star$ to $Y^\star \oplus (0\|\lfloor X^\star \rfloor_c)$ with label 0. Essentially, the p^{-1} queries behave similar to the p queries, and we obtain a directed edge-labelled graph. This is depicted in Fig. 4. Thus, overall we have a graph corresponding to all the queries. All the nodes computed during the hash computation (corresponding to the challenge queries) are called "H"-nodes and all the other nodes are called "P"-nodes. So, by definition, the number of H-nodes is σ_e, the total number of primitive calls required for the hash computation of all the challenge messages. The total number of P-nodes are bounded by $(q_p + \sigma_v)$, q_p being the total number of direct p and p^{-1} calls, and σ_v being the number of p calls used in the hash computation for the verification queries.

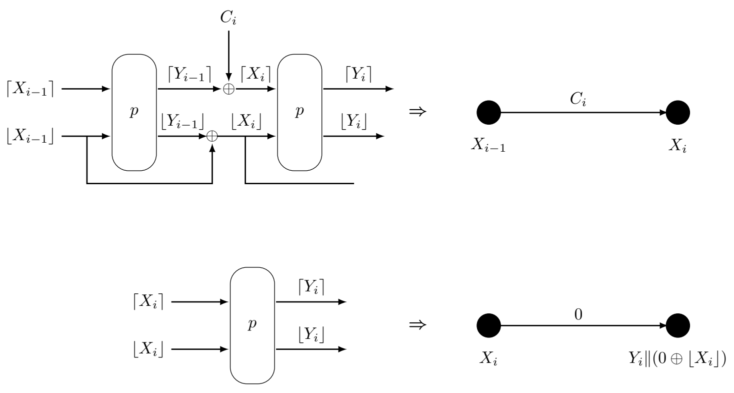

Fig. 4: The graph representation: challenge and forging queries (top), direct permutation queries (bottom).

Definition and Bounding the Probability of a Bad Graph. We call a collision occurs in two nodes if there capacity values are same. Now we call such a graph *bad* if there is a collision (i) among two starting "H" nodes, or (ii) due to a "H" node and a "P" node, (iii) between two "H" nodes. Now let us try to bound the probability that a graph is bad. For the first case, the initial state collision will reduce to a simple collision attack. This is due to the fact that the nonce and associated data are chosen by the adversary. Hence, this probability can be bounded by $q_p^2/2^{c'}$. For case (ii) and (iii), such a collision will occur with probability at most $\frac{1}{2^c - q_p}$, and the number of possible choice of H nodes and P nodes are σ_e and $(q_p + \sigma_v)$ respectively. Thus, the probability that a graph is *bad* can be bounded by $(\frac{q_p^2}{2^{c'}} + \frac{(q_p+\sigma_v)\sigma_e}{2^c} + \frac{\sigma_e^2}{2^c})$.
Bounding 2PI+ Security for A Good Graph. It is easy to see that if a graph is not *bad*, then we do not have any forgeries, except for random hash value matching, which can be bounded by $\frac{q_v}{2^{n-k}}$.
Combining everything together, the lemma follows. $\square$

Note that to extend the analysis for shorter challenge queries with $\ell \geq r'$ we need a domain separator at the end (adding 1 at the capacity). This is to resist an attack by guessing the random ciphertext and transferring a collision attack into a 2PI+ attack.

5 ISAP+: A Throughput-Efficient Variant of ISAP

In this section, we describe the ISAP+ family of NAEAD mode by instantiating EtHM with a sponge based PRF and the hybrid sponge hash and ultimately come up with the complete specification details of ISAP+.

5.1 Specification of ISAP+

Let n, k, r, r' and r_0 be five positive integers satisfying $1 < r, r', r_0 < n$, and IV_{KE}, IV_{KA} and IV_A be three $(n-k)$-bit binary numbers. We call the last three numbers as the initialization vectors. Let $c = n - r, c' = n - r'$ and $c_0 = n - r_0$. Let p be an n-bit public permutation. The authenticated encryption module of ISAP+ uses a secret key $K \in \{0,1\}^k$, receives a nonce $N \in \{0,1\}^k$, an associated data $A \in \{0,1\}^*$ and a message $M \in \{0,1\}^*$ as inputs, and returns a ciphertext $C \in \{0,1\}^{|M|}$ and a tag $T \in \{0,1\}^k$. The verified decryption module uses the same secret key K and receives a nonce $N \in \{0,1\}^k$, an associated data $A \in \{0,1\}^*$, a ciphertext $C \in \{0,1\}^*$ and a tag $T \in \{0,1\}^\tau$ as inputs. In case of successful verification, it returns a message $M \in \{0,1\}^{|C|}$. In case the verification fails, it returns $\perp$. Both the modules use a sub-module named re-keying (RK). The complete specification of ISAP+ is provided in Figure 5. The pictorial representation of the same is provided in Figures 6, 7, and 8.

Viewing ISAP+ as an Instantiation of EtHM: It is easy to that ISAP+ can be viewed as an instantiation of EtHM where the hash function H^p is given by the feed-forward variant of sponge hash as depicted in Figure 8 and the FIL-VOL keyed function F_k^p is described as follows:

– When $flag = 1$ (i.e., inside encryption module), F_k^p involves the rekeying function with $(n-k)$-bit output followed by p calls as depicted in Figures 6 and 7. The inputs are the nonce N, $flag = 1$ and a parameter ℓ which represents the message length. The number of p calls is equal to the number of r-bit message blocks.
– When $flag = 0$ (i.e., inside authentication module), F_k^p involves only the rekeying function with k-bit output as depicted in Figure 6. The inputs are the k most significant bits of the hash output, $flag = 0$ and a parameter $\ell = k$.

5.2 Design Rationale

In this section, we'll try to highlight and explain the main points regarding what motivated the design of EtHM, and in particular, ISAP+.

□ **Improved Rate for Ciphertext Processing in the Hash:** As we move from collision security to 2PI+ security at the ciphertext absorption phase of the authentication module, we achieve the same security with a smaller capacity size, which allows us to use a larger rate size for ciphertext absorption.

```
Algorithm ISAP+.AE_K(N, A, M)

1. $C \leftarrow$ ISAP+.Enc/Dec$(K, N, M)$
2. $T \leftarrow$ ISAP+.Auth$(K, N, A, C)$
3. return $(C, T)$
```

```
Algorithm ISAP+.Auth(K, N, A, C)

1. $A_1 \cdots A_a \xleftarrow{}\!\!\!\leftarrow_{r'} A$
2. if $|C| < r'$ then
3.     $C_1 \leftarrow C \| 10^{r'-|C|}$
4. else if $|C| = r'$ then
5.     $C_1 \leftarrow C$
6.     $C_2 \leftarrow 10^r$
7. else
8.     $C_1 \leftarrow \lceil C \rceil_{r'}$
9.     $C_2 \cdots C_\ell \xleftarrow{}\!\!\!\leftarrow_r \lfloor C \rfloor_{|C|-r'}$
10. $S \leftarrow N \| IV_A$
11. for $i = 1$ to $a$
12.     $S \leftarrow p(S) \oplus (A_i \| 0^{c'})$
13. $S \leftarrow p(S) \oplus (C_1 \| 0^{c'}) \oplus 0^{n-1}1$
14. for $i = 2$ to $\ell$
15.     $S \leftarrow p(S) \oplus (C_i \| 0^c) \oplus 0^r \| \lfloor S \rfloor_c$
16. $S \leftarrow p(S) \oplus 0^r \| \lfloor S \rfloor_c$
17. $S \leftarrow$ (ISAP+.RK$(K, \lceil S \rceil_k, 0, k)) \| \lfloor S \rfloor_{n-k}$
18. if $|C| < r'$ then
19.     $S \leftarrow S \oplus (0^{n-1} \| 1)$
20. return $T \leftarrow \lceil p(S) \rceil_k$
```

```
Algorithm ISAP+.VD_K(N, A, C, T)

1. $T' \leftarrow$ ISAP+.Auth$(K, N, A, C)$
2. if $T = T'$ then
3.     return ISAP+.Enc/Dec$(K, N, C)$
4. else
5.     return $\perp$
```

```
Algorithm ISAP+.Enc/Dec(K, N, X)

1. $X_1 \cdots X_\ell \xleftarrow{}\!\!\!\leftarrow_{r'} X$
2. $S \leftarrow N \|$(ISAP+.RK$(K, N, 1, n-k))$
3. for $i = 1$ to $\ell$
4.     $S \leftarrow p(S)$
5.     $Y_i \leftarrow \lceil S \rceil_{r'} \oplus X_i$
6. $Y \leftarrow \lceil Y_1 \| \cdots \| Y_\ell \rceil_{|X|}$
7. return $Y$
```

```
Algorithm ISAP+.RK(K, X, flag, z)

1. $IV \leftarrow (flag = 1)?\ IV_{KE} : IV_{KA}$
2. $X_1 \cdots X_w \leftarrow_{r_0} X$
3. $S \leftarrow p(K \| IV)$
4. for $i = 1$ to $(w-1)$
5.     $S \leftarrow p((\lceil S \rceil_{r_0} \oplus X_i) \| \lfloor S \rfloor_{n-r_0})$
6. $S \leftarrow p((\lceil S \rceil_{|X_w|} \oplus X_w) \| \lfloor S \rfloor_{n-|X_w|})$
7. return $\lceil S \rceil_z$
```

Fig. 5: Formal specification of the authenticated encryption and the verified decryption algorithms of ISAP+.

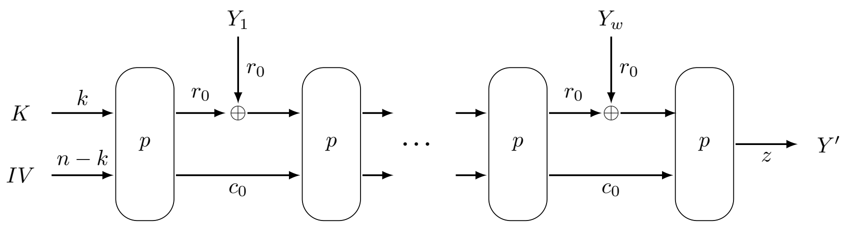

Fig. 6: Re-keying module of ISAP+ on a w-bit input Y.

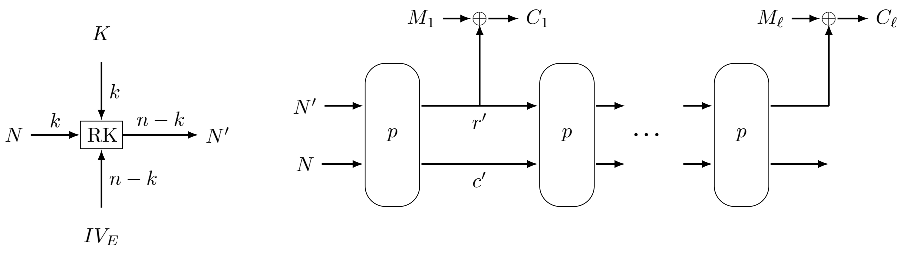

Fig. 7: Encryption module of ISAP+ for ℓ block message.

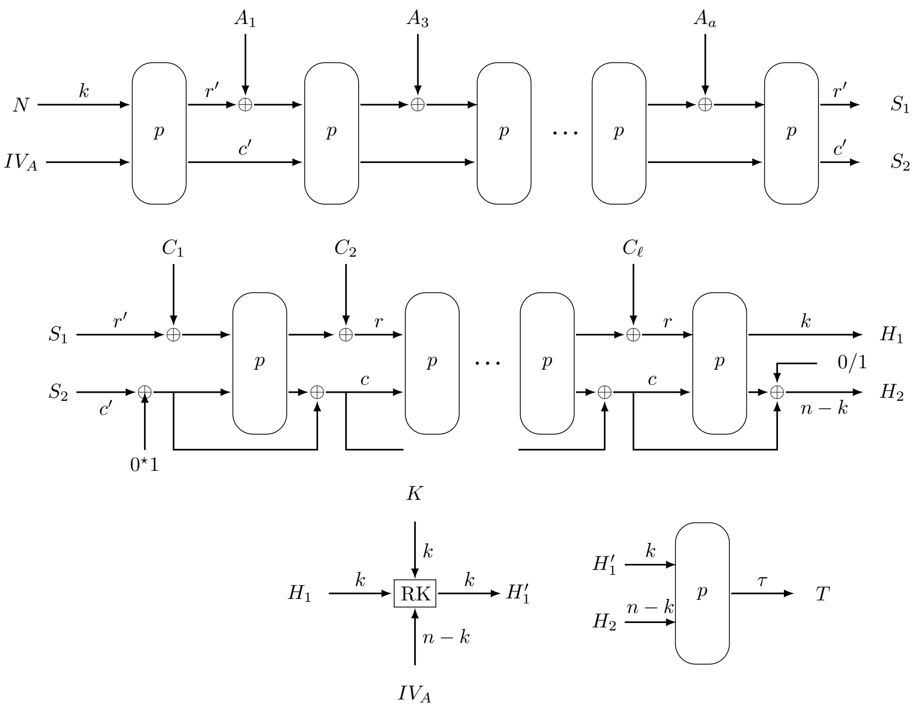

Fig. 8: Authentication module of ISAP+ for a block associated data and ℓ block message.

□ **Last Domain Separator:** The last domain separator is crucial to domain separate the short and long messages. Without this domain separator, we can have a forgery with one encryption query which consists of a message that is less than one block in length and the corresponding forging attempt which consists of more than one ciphertext blocks. As a result, a separator bit, applied to the capacity just before the last permutation call, allows us to differentiate these two cases, and ensure that the input to the last permutation is distinct for each of the two queries, which in turn prevents the attack.

5.3 Recommended Instantiations

We recommend one primary instance of ISAP+, denoted by ISAP+-A-128, which is instantiated with 12-round `ASCON`-p for all the permutation calls (i.e., each of s_H, s_B, s_E, and s_K is equal to 12). To showcase the efficiency of ISAP+, we've also implemented three other instances, comparable with the corresponding ISAP instances, as mentioned below in Table 3. Blue coloured entry is our primary recommendation.

Table 3: Recommended Parameter Values for ISAP+

Instances	Permutation	Bit Size of				No. of Rounds			
		n	r'	r	r_0	s_H	s_B	s_E	s_K
ISAP+-A-128	`ASCON`-p	320	64	128	1	12	12	12	12
ISAP+-A-128A	`ASCON`-p	320	64	128	1	12	1	6	12
ISAP+-K-128	`KECCAK`-p[400]	400	144	208	1	20	12	12	12
ISAP+-K-128A	`KECCAK`-p[400]	400	144	208	1	16	1	8	8

5.4 Security of ISAP+

In this subsection, we analyse the NAEAD security of ISAP+. We show that our design follows that paradigm, and hence we can adapt its security result, and the security of ISAP+ follows. Formally, we prove the following theorem.

Theorem 3 (NAEAD Security of ISAP+). *For all deterministic nonce-respecting non-repeating query making adversary $\mathcal{A}$ of ISAP+ which can make at most q_e encryption queries of a total of maximum σ_e blocks, q_v forging queries and q_p primitive queries to p and its inverse, the NAEAD advantage of $\mathcal{A}$ can be bounded by*

$$\mathbf{Adv}_{\mathsf{ISAP+}}^{NAEAD}(\mathcal{A}) \leq \frac{\sigma_e^2 + \sigma_e q_p + q_p^2}{2^{c'}} + \frac{\sigma_e^2 + \sigma_e q_p + \sigma_e \sigma_v}{2^c} + \frac{\binom{q_p}{k}}{2^{\tau(k-1)}} + \frac{\binom{q_p}{k}}{2^{(n-k)(k-1)}} + \frac{q q_p}{2^n} + \frac{2k q_v + q_p}{2^k} + \frac{q_p + q_v}{2^{n-k}} + \frac{q_v}{2^\tau}.$$

Proof. The proof follows directly from Theorem 2, as we bound the two terms $\mathbf{Adv}_F^{\mathsf{PRF}}(\mathcal{B}_1)$ and $\mathbf{Adv}_{\lfloor H \rfloor_c}^{\mathsf{2PI+}}(\mathcal{B}_2)$ for ISAP+.

– To bound the first term, we observe that the key can be randomly guesses with probability $\frac{q_p}{2^k}$. Also, the state after re-keying might match with an offline query with probability $\frac{q_p}{2^{n-k}}$, as the capacity part can be controlled. Otherwise the inputs of the outer sponge construction are fresh, and a collision can happen at some stage only if two construction queries have a full

state collision, or a construction query has a full state collision with a primitive query. The probability of the first case can be bounded by $\sigma_e^2/2^{c'}$ and the probability of the second case can be bounded by $\sigma_e q_p/2^{c'}$. Hence we achieve the overall PRF security of F as $\left(\frac{\sigma_e q_p+\sigma_e^2}{2^{c'}}+\frac{q_p}{2^k}+\frac{q_p}{2^{n-k}}\right)$. Further details can be found in [4].
– The second term, i.e., the 2PI+ security of the feed-forward based sponge hash can be bounded by $\left(\frac{q_p^2}{2^{c'}}+\frac{(q_p+\sigma_v)\sigma_e}{2^c}+\frac{\sigma_e^2}{2^c}+\frac{q_v}{2^{n-k}}\right)$ (See Lemma 2, Section 4.2).

□

Side-Channel Resistance. ISAP+ inherits its security against side-channel leakage directly from ISAP. In [18], the author have clearly mentioned that "There are no requirements on the implementation of the hash function H, since it processes only publicly known data." Following their argument, ISAP+ achieves same leakage resilience as it modifies only the hash function of ISAP and retains the rest of the design as it is. Accordingly, ISAP+ will provide a similar result on the leakage resilience bound as given in [19, Theorem 1].

6 Hardware Implementation Details

In this section, we provide the FPGA implementation detail of the ISAP+ family. All the hardware implementation codes are written in VHDL and are mapped on Virtex 7XC7V585T using Vivado 2020.2 as the underlying tool. The detail results are provided below.

6.1 Round Based Implementation of ASCON-p and KECCAK-p

In this section, we describe our round-based implementation of ASCON-p and adopted implementation of KECCAK-p[400]. The implementations are simply basic round based with n-bit datapath (n is the permutation size in bits). In case of ASCON-p, p receives a 320-bit (40-byte) input and processes it in 12 cycles (12 rounds in 12 clock cycles). Hence *cpb* for p is $40/12 = 3.33$. The circuit maintains a 320-bit internal state register which is initialised with the input and then gets updated after every round. It also maintains an additional 4-bit register **Round** to store the current round number. **Round** is initialized by 0 and is incremented by one after every round. After all the rounds are executed, **Round** is again initialised to 0. The architecture executes one round of p in one clock cycle and each round consists of 3 sequential sub modules AC, $SBox$, and Lin. The state is divided into 64 5-bit nibbles. The row representation of the state divides the state into five rows (each of the rows is known as a $Word$), each consisting of 64 bits. AC adds a round specific 1-byte constant to the third row of the intermediate state. It takes two inputs, the intermediate state and the current round number. Next, the $SBox$ module applies a non-linear 5-bit sbox to each of the nibbles of the state. The Lin module is used for linear diffusion of the state. This module adds different rotated copies of each $Word$ (horizontally, within each $Word$) to the corresponding $Word$. We directly adopt the KECCAK-p[400] round function implementation from [1] and implement the round based architecture on our own. The Virtex 7 results for both the implementations are provided in Table 4.

Table 4: FPGA Results for ASCON-p and KECCAK-p[400]

	Slice registers	LUTs	Slices	Clock Cycle Time (nS)	Frequency (MHZ)	Throughput (Gbps)
ASCON-p	328	937	282	2.5	416.67	11.1
KECCAK-p[400]	415	980	283	2.6	384.61	8.55

6.2 Comparison Between ISAP+ and ISAP Virtex 7 Results

We compare the hardware implementation results of ISAP+ and ISAP versions using our own implementation. We would like to point out that all our implementations follow a similar architecture and ignore the overheads associated with the NIST LWC hardware API. Also note that, the throughputs for all the versions have been computed for sufficiently long inputs and the clock cycle overheads have been ignored.

Below, we provide FPGA comparison results of ISAP+ and ISAP versions. We implement both ISAP+ and ISAP using VHDL and mapped the implementation on Virtex 7XC7V585T (Vivado 2020.2). We use the same RTL approach and a basic iterative type architecture. We use a common hardware architecture for all the versions of ISAP and ISAP+. Note that we provide our own implementation for ASCON-p (as our main recommendation is based on ASCON-p), and we use the reference round function implementation of KECCAK-p[400] from [1].

The hardware implementation results for the four versions of ISAP+ is presented in Table 5. We also report our hardware implementation results for the four versions of ISAP. We implement them on the same platform using the same approach. The detailed results are described in Table 6 below. The result depicts that all the ISAP+ versions are better in throughput than the corresponding ISAP versions which in turn depicts that ISAP+ versions remain significantly better than ISAP versions with respect to throughput/area metric. Blue coloured entry in Table 5 is our primary recommendation.

Table 5: FPGA Results for ISAP+ versions.

Versions	Slice Registers	LUTs	Slices	Frequency (MHZ)	Encryption Throughput (Gbps)	Authentication Throughput (Gbps)
ISAP+-A-128	1081	1742	507	384.16	2.05	3.07
ISAP+-K-128	1423	2245	613	300	3.60	2.64
ISAP+-A-128A	1070	1728	501	384.16	4.09	3.07
ISAP+-K-128A	1412	2231	605	300	5.40	4.40

Table 6: FPGA Results for ISAP Versions

Versions	Slice Registers	LUTs	Slices	Frequency (MHZ)	Encryption Throughput (Gbps)	Authentication Throughput (Gbps)
ISAP-A-128	818	1550	451	400	2.13	2.13
ISAP-K-128	1143	1932	582	300	3.60	2.16
ISAP-A-128A	810	1539	449	400	4.27	2.13
ISAP-K-128A	1131	1850	574	300	5.40	2.70

7 Conclusion

In this paper, we have proposed a generic framework for a permutation-based EtHM type NAEAD mode using a PRF and an unkeyed hash function with 2PI+ security. We have shown that ISAP follows the framework EtHM and hence it's security boils down to the 2PI+ security of the underlying hash function. We propose a feed-forward variant of the sponge hash function with improved security and use it to design a new variant of ISAP that achieves improved security, and that in turn improves the throughput of the construction. Designing some new hash functions with better 2PI+ security and improving the security or throughput of the mode instantiated with the newly designed hash seems to be a challenging open problem.

Acknowledgement. The authors would like to thank all the anonymous reviewers for their valuable comments and suggestions. Cuauhtemoc Mancillas López is partially supported by the Cryptography Research Center of the Technology Innovation Institute (TII), Abu Dhabi (UAE), under the TII- Cuauhtemoc project.

References

1. `https://keccak.team/hardware.html`.
2. National Institute of Standards and Technology. FIPS PUB 202: SHA-3Standard: Permutation-based hash and extendable-output functions. . Federal Information Processing Standards Publication 202, U.S. Department ofCommerce, 8 2015.
3. Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Florian Mendel, Bart Mennink, Nicky Mouha, Qingju Wang, and Kan Yasuda. PRIMATEs v1.02. Submission to CAESAR. 2016. `https://competitions.cr.yp.to/round2/primatesv102.pdf`.
4. 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 *Lecture Notes in Computer Science*, pages 364–384. Springer, 2015.
5. 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. Cryptogr. Eng.*, 4(3):157–171, 2014.

6. Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000.
7. Daniel J. Bernstein. ChaCha, a variant of Salsa20. 2008.
8. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge functions, 2007. Ecrypt Hash Workshop 2007.
9. 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 *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008.
10. Guido Bertoni, Michaël Peeters Joan Daemen, Gilles Van Assche, and Ronny Van Keer. Ketje v2. Submission to CAESAR. 2016. `https://competitions.cr.yp.to/round3/ketjev2.pdf`.
11. Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski Jr., editor, *CRYPTO '97, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.
12. Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *EUROCRYPT 1997*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.
13. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
14. Christoph Dobraunig and Maria Eichlseder and Florian Mendel and Martin Schläffer. Ascon v1.2. Submission to NIST Lightweight Cryptography, 2019. 2019. `https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/ascon-spec-final.pdf`.
15. Christoph Dobraunig and Maria Eichlseder and Stefan Mangard and Florian Mendel and Bart Mennink and Robert Primas and Thomas Unterluggauer. ISAP v2.0. Submission to NIST. 2019. `https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/ISAP-spec.pdf`.
16. CAESAR Committee. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. `http://competitions.cr.yp.to/caesar.html/`.
17. Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.
18. Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, Bart Mennink, Robert Primas, and Thomas Unterluggauer. Isap v2.0. `https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/isap-spec-final.pdf`.
19. Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, Bart Mennink, Robert Primas, and Thomas Unterluggauer. Isap v2.0. *IACR Trans. Symmetric Cryptol.*, 2020(S1):390–416, 2020.
20. Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, and Thomas Unterluggauer. ISAP - towards side-channel secure authenticated encryption. *IACR Trans. Symmetric Cryptol.*, 2017(1):80–105, 2017.
21. Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. Submission to CAESAR. 2016. `https://competitions.cr.yp.to/round3/asconv12.pdf`.

22. Benoît Gérard, Vincent Grosso, María Naya-Plasencia, and François-Xavier Standaert. Block ciphers that are easier to mask: How far can we go? In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013. Proceedings*, volume 8086 of *Lecture Notes in Computer Science*, pages 383–399. Springer, 2013.
23. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.
24. Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems, CHES 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.
25. Vincent Grosso, Gaëtan Leurent, Francois-Xavier Standaert, Kerem Varici, Anthony Journault, Francois Durvaux, Lubos Gaspar, and Stéphanie Kerckhof. SCREAM Side-Channel Resistant Authenticated Encryption with Masking. Submission to CAESAR. 2015. `https://competitions.cr.yp.to/round2/screamv3.pdf`.
26. Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche. The Keccak reference (version 3.0). 2011. `https://keccak.team/files/Keccak-reference-3.0.pdf`.
27. Joan Daemen, Michaël Peeters, Gilles Van Assche, and Vincent Rijmen. The NOEKEON block cipher, 2000. Nessie Proposal. 2020. `https://competitions.cr.yp.to/round3/acornv3.pdf`.
28. Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *CRYPTO 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
29. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
30. 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 *Lecture Notes in Computer Science*, pages 115–132. Springer, 2011.
31. 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, Proceedings*, volume 6055 of *Lecture Notes in Computer Science*, pages 279–296. Springer, 2010.
32. NIST. Lightweight cryptography. `https://csrc.nist.gov/Projects/Lightweight-Cryptography`.
33. Jacques Patarin. The "coefficients h" technique. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography*, pages 328–345, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
34. Gilles Piret, Thomas Roche, and Claude Carlet. PICARO - A block cipher allowing efficient higher-order side-channel resistance. In Feng Bao, Pierangela Samarati, and Jianying Zhou, editors, *ACNS 2012. Proceedings*, volume 7341 of *Lecture Notes in Computer Science*, pages 311–328. Springer, 2012.