A New Stand-Alone MAC Construct Called SMAC

Dachao Wang[1], Alexander Maximov[2], Patrik Ekdahl[2] and Thomas Johansson[1]

[1] Dept. of Electrical and Information Technology, Lund University, Lund, Sweden
{dachao.wang,thomas.johansson}@eit.lth.se
[2] Ericsson Research, Lund, Sweden
{alexander.maximov,patrik.ekdahl}@ericsson.com

Abstract. In this paper, we present a new efficient stand-alone MAC construct named SMAC, based on processing using the Finite State Machine (FSM) part of the stream cipher family SNOW, which in turn uses the AES round function. It offers a combination of very high speed in software and hardware with a truncatable tag. Three concrete base versions of SMAC are proposed, each offering a different security level. SMAC can also be directly integrated with an external ciphering engine in an AEAD mode. Every design decision is thoroughly justified and supported by the results of our cryptanalysis and simulations. Additionally, we introduce an aggregated mode version, SMAC-1×n, in which software performance reaches up to 925 Gbps (around 0.038 cycles per byte) for long messages in a single thread. To the best of our knowledge, SMAC achieves a record-breaking software performance compared to all known MAC engines.

Keywords: MAC · SNOW · AES

1 Introduction

A Message Authentication Code (MAC) is a standard symmetric primitive for two parties that share a secret key to verify that a received message originates from the sending party and that it was not modified by an attacker when sent on a possibly insecure channel.

Traditionally, most existing MACs are built either from a block cipher or from cryptographic hash functions. Common examples of constructs based on block ciphers are CBC-MAC and CMAC [IK03]. A common hash-based construct is HMAC [BCK96]. Another common direction is to use universal hash functions (or equivalently unconditionally secure authentication codes [BJKS94]) as a basis for constructs, resulting in schemes like UMAC [BHK+99], Poly1305-AES [Ber05] and GMAC [Dwo07].

The widespread use of the AES encryption standard turned CPU vendors to introduce an AES-NI set of instructions that *de facto* became a standard component. In order to boost software performance, AES-NI is used as a 128-bit nonlinear S-box in many modern cryptographic designs, for example, the ciphers [WP14, EJMY19, SLN+21] and the hash and MAC algorithms [BÖS11, DR05, JN16, BBL+24], etc.

In this paper, we present a new efficient stand-alone class of MAC constructs, called SMAC, that is based on processing using the Finite State Machine (FSM) part of the stream cipher family SNOW, which in turn uses the AES round function. It offers a combination of very *high speed in software*, *efficiency in hardware*, *truncatable MAC*, and a decent *robustness in a nonce-misuse scenario*. We also introduce an *aggregated mode*, SMAC-1×n, built on the base version SMAC-1, where speed reaches high performance in

the interval 0.059-0.038 cycles per byte (cpb) in software in a single thread. The latter is a clear way to utilise the parallelisation capabilities of modern CPUs and can be similarly applied to the other two base variants SMAC-3/4 and SMAC-1/2.

The core of SMAC consists of only three 128-bit registers that are iterated through the use of two AES round functions and an important byte-wise permutation on one register. A block of the message enters as input in the processing of each register.

The design and security analysis considers Maximum Degree Monomial (MDM) tests and cube attacks, Time-Memory Trade-Off (TMTO), Guess-and-Determine (GnD), nonce-misuse, Key/IV/message differentials for MAC forgery, and study clustering effects of differential trails as well as the number of active S-boxes for different choices of a particular permutation through constraint programming (CP) modelling tools. Similar to [BBL+24], the security analysis in this paper is based on simulations, instead of a formal security proof, and every design choice is justified and supported by these results.

A valid question is how SMAC compares to existing state-of-the-art designs. In June 2024, two new ultra-fast MAC constructs based on AES-NI both offering 128-bit security, called LeMac and PetitMac were published in ToSC [BBL+24]. The paper provides a comprehensive overview of existing solutions, state of the art, and why these two new MAC constructs are advantageous over prior work. Therefore, it seems justified in this paper to provide a comparison of SMAC vs LeMac and PetitMac and demonstrate the competitiveness of SMAC compared to these most recent designs.

PetitMac processes one 128-bit message block per round with the help of two *sequential* AES-NI calls, plus some other XOR operations. PetitMAC has six 128-bit registers as its internal state. The SMAC-1 construct presented in this paper also uses two AES-NI calls to process a single message block, but these two calls are *independent* of each other which allows the SMAC design to run much faster in software, which is a clear advantage. Also, the state of SMAC-1 is only three 128-bit registers – half of the state of PetitMac. There is, however, a minor drawback of SMAC-1 that the claimed security level is 118 bits for tags larger than 118 bits, and full security for shorter tags – this is slightly less than the 128 bits with PetitMac. However, we believe it is a minor issue for real use cases as many applications require short tags, and this can be compared to e.g. GHASH used in GCM with a 128-bit tag, which provides only 96 bits of security while the tag is not truncatable (the security degrades along the size of the tag). Furthermore, the security level of 118 bits represents a lower bound following the results of cluster analysis under worst-case assumptions, and the real security level might actually be higher; we leave this for future research.

LeMac is the design that aims to benefit from parallel calls to multiple AES cores implemented in modern CPUs, as well as leveraging on interleaving effects and/or wider ZMM registers from AVX-512. A single call to LeMac processes 4 message blocks by having 8 calls to the AES round, and has the state of 12 128-bit registers, with a maximum performance of 0.068 cpb, or 514 Gigabits per second (Gbps). A comparable instance to LeMac would be the aggregated mode version SMAC-1 $\times$ 4 which also processes 4 message blocks per a call with 8 calls to AES round. The measured speed of SMAC-1 $\times$ 4 is up to 0.059 cpb (590 Gbps), which is competitive.

An advantage of SMAC-1 $\times$ n is that n here can be any number from 1 to 16, thus representing a class in the SMAC family. For instance, SMAC-1 $\times$ 8 achieves a performance of 925 Gbps, which is twice as fast as LeMac. That property seems harder to achieve with the Le/PetitMac approach as it would involve a new construct requiring additional simulations and analysis. Also, if a higher security level is needed, one can pick SMAC-1/2 (or SMAC-3/4) and use it in the aggregated mode SMAC-1/2 $\times$ n similarly to SMAC-1 $\times$ n. Finally, SMAC is supplied with fast initialisation and finalisation procedures and does not require derivation and storage of e.g. round keys.

A key distinction of SMAC compared to previous AES-NI-based MAC designs is its

use of a byte-wise permutation. This enables the creation of secure constructs with a small state and yet parallel AES-NI calls per round, where the small state facilitates an efficient aggregated mode. These advantages bring SMAC much closer to the limits of current software performance capabilities.

The paper is organised as follows. In Section 2 we present the details of the SMAC design, its different versions and our security claims. Section 3 provides a thorough security analysis of the design, focusing particularly on differential attacks, while also considering other potential attack vectors. The conclusions drawn from these analyses are also transferred to design justifications, explaining why parameters are chosen as they are. In particular, the search for the most optimal byte-wise permutation used in the design is given significant attention. Section 4 offers a short description of an aggregated version of SMAC along with a brief analysis. Section 5 provides the software evaluation of different SMAC versions.

2 The SMAC construct

We propose a new MAC engine called SMAC. It is derived from the finite state machine part of the SNOW family [EJMY19], as depicted in Figure 1. The SMAC construct has three 128-bit internal state registers $(A1, A2, A3)$ and their values at time t are denoted by $(A1^t, A2^t, A3^t)$. Given a 128-bit wide message word M^t at time t, the internal registers are updated by the compression function Π as follows:

$$(A1^{t+1}, A2^{t+1}, A3^{t+1}) \leftarrow \Pi(A1^t, A2^t, A3^t, M^t) \;:=\; \begin{cases} A1^{t+1} = \sigma(A2^t \oplus A3^t \oplus M^t) \\ A2^{t+1} = AES_R(A1^t, M^t) \\ A3^{t+1} = AES_R(A2^t, M^t) \end{cases}$$

where σ is a fixed 16-byte permutation, and $AES_R(X, K)$ is the AES round function. A

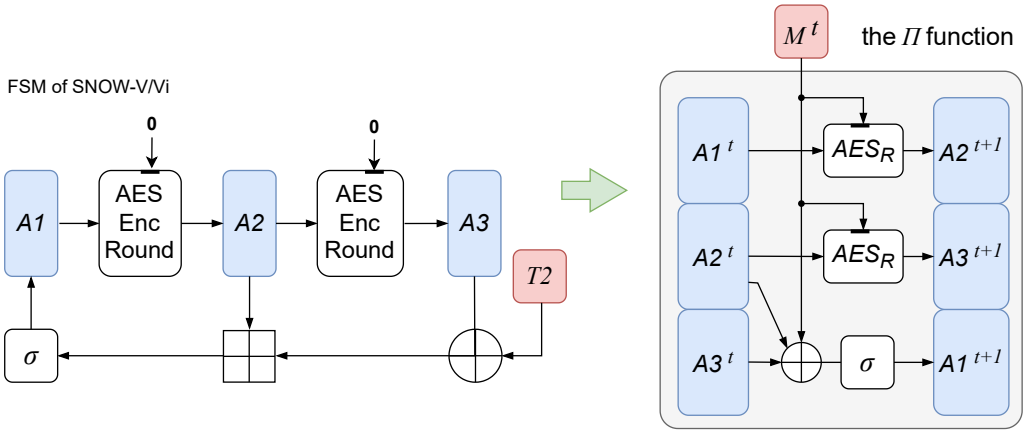

Figure 1: SMAC compression function Π, derived from the FSM part of SNOW-V/Vi.

general way of usage of the compression function Π in the SMAC framework is shown in Figure 2, and consists of three phases:

- **Initialisation phase.** In the initialisation phase, the three 128-bit registers $A1, A2, A3$ are populated with the key material and any other relevant domain separation parameters, for example a nonce and the MAC tag size. Next, the compression function Π is iterated d times using a fixed $M = \mathbf{1}^\star$ (to be defined later), to

transition the initial state into a pseudo-random state. The internal state at the end of the initialisation phase is denoted by $(A1^t, A2^t, A3^t), t = 0$.

- **Compression phase.** The next n clocks are used to compress the sequence of n 128-bit message blocks $M^0, \ldots, M^{n-1}$, where the last message block should contain the actual length of the message in bits. The internal state after the compression phase is denoted by $(A1^t, A2^t, A3^t), t = n$.
- **Finalisation phase.** Before the MAC tag is produced, the SMAC engine does d dummy calls to the compression function with $M = \mathbf{1}^\star$, similarly to the initialisation phase. The output MAC tag is extracted from the state $(A1^t, A2^t, A3^t), t = n + d$, by taking the required number of bits.

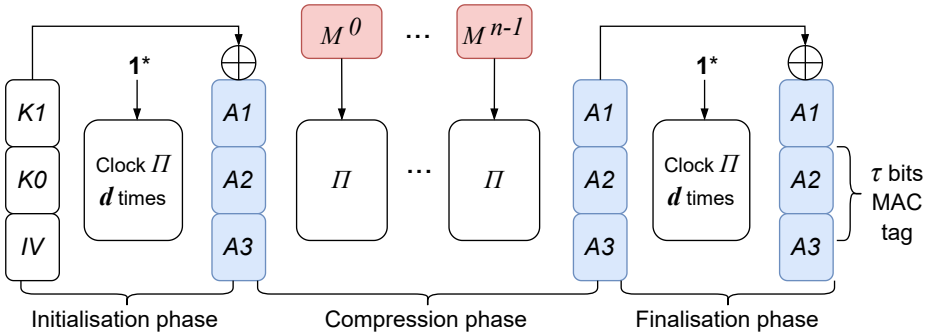

Figure 2: A general usage model of the SMAC framework.

2.1 Detailed description

The description is byte oriented and we will denote an array of bytes of length l by $\{\mathbb{N}_8\}^l$, where $\mathbb{N}_k$ denotes the natural numbers representable using k bits. The elements in an array $A \in \{\mathbb{N}_8\}^l$ are referenced by $A[0], A[1], \ldots, A[l-1]$, where $A[0]$ is the first element in the array and $A[l-1]$ is the last. An assignment of an array $A = B$ is done element by element, as is the XOR (also denoted by $\oplus$) of two arrays $C = A \oplus B$, where then $C[0] = A[0] \oplus B[0]$, et cetera. The registers are considered byte arrays $A1, A2, A3 \in \{\mathbb{N}_8\}^{16}$.

The concatenation of two arrays A, B is denoted by $C = A \parallel B$ and the result C will carry the elements of A in its first positions and the elements of B in its last positions. Let $\sigma(\cdot) : \{\mathbb{N}_8\}^{16} \to \{\mathbb{N}_8\}^{16}$ denote a byte permutation of an array of length 16. A specific permutation is defined as $\sigma = [\pi_0, \pi_1, \ldots, \pi_{15}], \pi_k \in [0 \ldots 15], \pi_i \neq \pi_j \; \forall (i \neq j)$. This should be interpreted as the element at index π_0 is moved to position 0, the element at index π_1 is moved to position 1, and so on. For example, $\sigma(A)$ will result in the permuted array $B = \sigma(A) = \{A[\pi_0], A[\pi_1], \ldots, A[\pi_{15}]\}$. Furthermore, let

$$\mathbf{1}^\star = \{1, 0, 0, \ldots, 0\}$$

denote the array of 16 bytes with a single one in the first position and zeros in the rest. $M = \mathbf{1}^\star$ is the fixed constant value fed into Π during initialisation and finalisation phases. A single instance of the AES round function is denoted by $AES_R(X, K) : (\{\mathbb{N}_8\}^{16}, \{\mathbb{N}_8\}^{16}) \to \{\mathbb{N}_8\}^{16}$, and defined as $AES_R(X, K) := \texttt{MixColumns}(\texttt{ShiftRows}(\texttt{SubBytes}(X))) \oplus K$. In subsequent sections of this paper, we will also use the notation $L \cdot X = \texttt{MixColumn}(X)$, $\pi \cdot X = \texttt{ShiftRows}(X)$, and $S(X) = \texttt{SubBytes}(X)$, so that the AES round can be rewritten in a shorter form as

$$AES_R(X, K) = L\pi S(X) \oplus K$$

The mapping between a byte array and the AES state X is done in the usual way as defined in [oST01]. We can now formally define the compression function Π in Algorithm 1. The initialisation and finalisation phases are identical in the proposed framework and defined in Algorithm 2.

Algorithm 1 $\Pi : (\{\mathbb{N}_8\}^{16}, \{\mathbb{N}_8\}^{16}, \{\mathbb{N}_8\}^{16}, \{\mathbb{N}_8\}^{16}) \rightarrow (\{\mathbb{N}_8\}^{16}, \{\mathbb{N}_8\}^{16}, \{\mathbb{N}_8\}^{16})$

function $\Pi(A1, A2, A3, M) \rightarrow (A1', A2', A3')$
  $A1' = \sigma(A2 \oplus A3 \oplus M)$
  $A2' = AES_R(A1, M)$
  $A3' = AES_R(A2, M)$

Algorithm 2 `InitFinal` $: (\{\mathbb{N}_8\}^{16}, \{\mathbb{N}_8\}^{16}, \{\mathbb{N}_8\}^{16}) \rightarrow (\{\mathbb{N}_8\}^{16}, \{\mathbb{N}_8\}^{16}, \{\mathbb{N}_8\}^{16})$

function `InitFinal`$(A1, A2, A3) \rightarrow (A1', A2', A3')$
  $(X1, X2, X3) = (A1, A2, A3)$
  for d times **do** ▷ In this specification $d = 9$
    $(A1, A2, A3) = \Pi(A1, A2, A3, \mathbf{1}^\star)$
  $(A1', A2', A3') = (A1, A2, A3) \oplus (X1, X2, X3)$

Since Π given M is an invertible function, we prevent back-tracking of the state from the exposed MAC tag by adding the starting state of the function to the ending state. This is also done during the initialisation phase to achieve an instantiation of the FP(1) property introduced in [HK18].

2.2 Three base instances SMAC-1, SMAC-3/4, and SMAC-1/2

While the previous subsection provides a more general description, we need some additional specifications to instantiate an implementable algorithm. In this section we specify three concrete base instances called SMAC-1, SMAC-3/4, and SMAC-1/2. They work as stand-alone integrity algorithms that provide truncatable tags of size τ bits, where the upper limit is $\tau \leq \{128, 160, 256\}$ bits for these three instances, respectively.

The first instance, SMAC-1, has a lower security level but processes a new message block for each round during the compression phase. The second variant, SMAC-3/4, has a higher security level but only processes three message blocks every 3 out of 4 compression rounds. It does so by running the compression function Π with $M = \mathbf{1}^\star$ every fourth round of the compression phase. The resulting rate is 3/4 of the rate of SMAC-1, neglecting the identical initialisation and finalisation phases. The third variant is the half-rate SMAC-1/2, where every second clock of the compression phase is the dummy clock with $M = \mathbf{1}^\star$.

All three instances take a 256-bit key $K \in \{\mathbb{N}_8\}^{32}$ and a 128-bit domain separation value $IV \in \{\mathbb{N}_8\}^{16}$ as inputs. Two 128-bit halves of the key to be referred as $K0$ and $K1$, i.e. $K = (K0 \parallel K1)$. If the original key is shorter then the 256-bit K is constructed from the original shorter key by extending it to 256 bits with zeroes. Exactly how the domain separation is to be done is left for the user of the algorithms, but separating different key and tag sizes together with a nonce should probably be considered. The output tag of SMAC-{1, 3/4, 1/2} is confined to a maximum of {16, 20, 32} bytes, and corresponds to the first bytes of the registers $(A2 \parallel A3)$ after the finalisation phase.

We define the SMAC versions such that they are directly suitable for use in AEAD mode with an external cipher. For this, we assume the message is comprised of two parts – one part with associated data (AD), and one part with ciphertext data. Let $\mathcal{A} \in \{\mathbb{N}_8\}^{L_A}$ be the plaintext AD of length L_A bytes, and let $\mathcal{C} \in \{\mathbb{N}_8\}^{L_C}$ be the ciphertext data of length L_C bytes. We form the input message to SMAC-{1, 3/4, 1/2} by firstly pad $\mathcal{A}$ and

$\mathcal{C}$ to 16 bytes boundaries by inserting 0:s. Then the ciphertext array is concatenated to the end of the AD array, followed by a 16 byte block consisting of the lengths in bits of the messages. The conversion from integer to byte array is done in little endianness style with the least significant byte in the first array element, and denoted by the conversion function $\texttt{LittleEndian64}(n)$. Finally, we fix the number of rounds during the initialisation and finalisation phases (Algorithm 2) to $d = 9$. In Algorithm 3 we provide the complete description of SMAC-{1, 3/4, 1/2}.

Algorithm 3 SMAC-r : $(\{\mathbb{N}_8\}^{32}, \{\mathbb{N}_8\}^{16}, \{\mathbb{N}_8\}^{L_A}, \{\mathbb{N}_8\}^{L_C}) \rightarrow \{\mathbb{N}_8\}^{16/20}$

1: **function** SMAC-{1, 3/4, 1/2}$(K, IV, \mathcal{A}, \mathcal{C}) \rightarrow Tag$
2: Assign a 16-byte block $\mathcal{L} = \texttt{LittleEndian64}(8 \cdot L_A) \parallel \texttt{LittleEndian64}(8 \cdot L_C)$
3: Add zeroes to $\mathcal{A}$ and $\mathcal{C}$ to align with 16-byte blocks, resulting in $\mathcal{A}^*$ and $\mathcal{C}^*$
4: $\mathcal{M} = \mathcal{A}^* \parallel \mathcal{C}^* \parallel \mathcal{L}$, and $L_{\mathcal{M}} = 16 \cdot (\lceil L_A/16 \rceil + \lceil L_C/16 \rceil + 1)$
5: Divide $\mathcal{M}$ into $L_{\mathcal{M}}/16$ sub-blocks M^i of size 16 bytes, index starts from $i = 0$
6: $(A1, A2, A3) = (K1, K0, IV)$ ▷ Note, the lower half of K is loaded into $A2$
7: $(A1, A2, A3) = \texttt{InitFinal}(A1, A2, A3)$
8: **for** all sub-block M^i in $\mathcal{M}$ **do**
9: $(A1, A2, A3) = \Pi(A1, A2, A3, M^i)$
10: **if** (SMAC-3/4 $\wedge$ $(i \equiv 2 \bmod 3)$) **or** (SMAC-1/2) **then**
11: $(A1, A2, A3) = \Pi(A1, A2, A3, \mathbf{1}^\star)$
12: $(A1, A2, A3) = \texttt{InitFinal}(A1, A2, A3)$
13: $Tag = (A2 \parallel A3)_\tau$ ▷ First pick tag bits from $A2$, then from $A3$ if $\tau > 128$

The three instances use distinct permutations σ, which are defined as follows:

$$\begin{aligned}
\sigma_1 &= \{0, 7, 14, 11, 4, 13, 10, 1, 8, 15, 6, 3, 12, 5, 2, 9\} \quad \text{for SMAC-1} \\
\sigma_{42} &= \{7, 14, 15, 10, 12, 13, 3, 0, 4, 6, 1, 5, 8, 11, 2, 9\} \quad \text{for SMAC-3/4} \\
\sigma_{61} &= \{0, 11, 7, 14, 6, 4, 1, 15, 9, 3, 8, 5, 13, 2, 10, 12\} \quad \text{for SMAC-1/2}
\end{aligned}$$

SMAC-{1, 3/4, 1/2} only admit byte oriented inputs and hence we multiply the array length by 8. If bit oriented inputs are needed, simply provide the total number of bits as input and use those values in line 2. The input data arrays need to be byte aligned in any case. The length encoding puts a restriction on the length of the plaintext and ciphertext messages to be maximum $2^{64} - 1$ bits each.

For the tag extraction, we assume τ to be a constant parameter representing the size of the Tag in bits; recall that the tag size τ is at most 128, 160 or 256 bits, depending on the SMAC variant. The extraction from the state is primarily done from register $A2$ where the bytes of Tag are assigned by the corresponding indices of $A2$. If $\tau > 128$ the Tag array is appended by bytes from register $A3$, starting with the first position.

2.3 Security claims and limitations

In the context of the current specification of SMAC-{1, 3/4, 1/2}, with $d = 9$ for both the initialisation and finalisation phases, we summarise security claims and limitations as given in Table 1.

The nonce-misuse using a sender oracle scenario discussed in Section 3.13 is highly theoretical since if the goal is to send some selected malicious messages to the receiver, the direct solution would be to ask the oracle for the correct tag, instead of using 2^{59} queries in order to recover the state. As will be discussed in Sections 3.7 and 3.9, a state recovery does not directly translate into a key recovery so the state is only valid for that particular (K, IV). The receiver oracle is more plausible in a practical scenario, since if the protocol does not include some replay protection, the receiver can indeed be used to verify correctly

Table 1: Security claims and limitations. Q=Queries, T=Time, C=Time/Memory/Data.

Security and limitation aspects	SMAC-1	SMAC-3/4	SMAC-1/2
Original key size in bits, κ	$\kappa \leq 256$	$\kappa \leq 256$	$\kappa \leq 256$
Truncated tag size in bits, τ	$\tau \in [12..128]$	$\tau \in [12..160]$	$\tau \in [12..256]$
Maximum length of AD and ciphertext in bits	$2^{64}-1$	$2^{64}-1$	$2^{64}-1$
Maximum number of messages with the same key	2^{64}	2^{64}	2^{64}
Security level against a single message forgery in a nonce-respecting setting, in bits (Sec.3.3)	$\geq \min\{\kappa, \tau, 118\}$	$\geq \min\{\kappa, \tau, 152\}$	$\geq \min\{\kappa, \tau, 252\}$
Security level against input/output differentials in `InitFinal`, in bits (Sec.3.11)	$\geq \min\{\kappa, 56 \cdot 6\}$	$\geq \min\{\kappa, 49 \cdot 6\}$	$\geq \min\{\kappa, 54 \cdot 6\}$
GnD to recover the key from a known single state and IV (Sec.3.9)	$T = O(2^{\kappa})$	$T = O(2^{\kappa})$	$T = O(2^{\kappa})$
GnD to recover the state from a known tag(s) when (K, IV) is fixed (Sec.3.9)	$T = O(2^{384})$	$T = O(2^{384})$	$T = O(2^{384})$
TMTO to recover the state with multiple known tags (Sec.3.7)	$C = O(2^{192})$	$C = O(2^{192})$	$C = O(2^{192})$
TMTO to recover the key with a fixed IV and multiple known tags (Sec.3.7)	$C = O(2^{\kappa/2})$	$C = O(2^{\kappa/2})$	$C = O(2^{\kappa/2})$
State recovery with nonce-misuse queries to the receiver oracle (Sec.3.13)	$Q \geq O(2^{59+\tau})$	$Q \geq O(2^{76+\tau})$	$Q \geq O(2^{126+\tau})$

guessed state collisions. From the table we conclude that even if the smallest allowed tag size $\tau = 12$ is used, the complexity of a receiver nonce-misuse attack is well above the allowed number of messages to be MACed for that key.

As a sound security practice, we recommend that each pair (K, IV) be used only once by the sender. In order to prevent receiver's side nonce-misuse attacks in protocols that lack replay protection, we suggest introducing a counter of the number of verification requests per key. If this counter reaches the maximum allowed value (note the last row of Table 1), the receiver should invalidate the key and, for instance, request a key renegotiation. In scenarios with multiple independent receivers, say m, the upper limit of the counter should be divided equally among the m receivers.

3 Design justifications and security analysis

3.1 Differential forgery attacks

For the sake of terminology, we call bit-level differential trails as *bit-trail*s which depict concrete differential trails in practical attacks. We also name a byte-level trail as a *byte-trail* that indicates whether each differential byte value is zero or not. This type of trail is good for e.g. counting the minimum number of active S-boxes. A bit-differential message is denoted by ΔM, and its corresponding byte-differential is denoted by $\mu(\Delta M)$.

One of the main security concerns in such constructs comes from the second preimage resistance, where an attacker given the first message M tries to construct a new message M' such that the resulting tag would coincide. In this scenario, the attacker may query the verification oracle to check if the tags for M and M' collide. Here we have two possibilities: either two tags of size τ bits would collide by chance with probability $2^{-\tau}$, or the attacker carefully selects $\Delta M = M \oplus M'$ such that the internal state would collide with probability p significantly larger than $2^{-\tau}$.

A standard approach to estimate the state collision probability p is to find a differential byte-trail $\mu(\Delta M)$ with the minimum number s of active S-boxes, then p can be upper bounded by $p \leq 2^{-6s}$, since for the Rijndael S-box we have $\forall \delta_x, \delta_y : \Pr\{S(x) \oplus S(x \oplus \delta_x) = \delta_y\} \leq 2^{-6}$. In this estimate the basic assumption is that each of active differential S-boxes would theoretically have a corresponding binary-trail with the maximum probability 2^{-6},

which could not always be the case and such an optimal binary-trail may not exist, pushing the actual p even lower. On the other hand, it is not always guaranteed that a byte-trail would have a corresponding binary-trail with a nonzero probability. However, this estimate was adopted by e.g. designers of AEGIS, Rocca, LeMac, PetitMac, etc.

Practically speaking, there are many well-known tools and methods for the search of a minimum byte-trail, such as using MILP models [MWGP12], CP tools [SGL+17], and custom programming like Matsui-style search [Mat95] – in our case, we also programmed our own search in C++ for e.g. the case $\Delta t = 3$, briefly described in Appendix B. Such techniques are well-studied and have been utilised to analyse many other cryptographic primitives, see e.g. [BBL+24, SLN+21] for more details.

As an example, one can easily derive the state expressions after 3 clocks for $\Delta t = 3$, where we introduce additional substitutions in order to make the expressions shorter $x = \sigma(L\pi S(A1^t) \oplus L\pi S(A2^t))$, $y = \sigma(A2^t \oplus A3^t)$, and $z = L\pi S(A1^t)$:

$$\begin{cases} A1^{t+3} = \sigma(L\pi S(y \oplus \sigma M^t) \oplus L\pi S(z \oplus M^t) \oplus M^{t+2}) \\ A2^{t+3} = L\pi S(x \oplus \sigma M^{t+1}) \oplus M^{t+2} \\ A3^{t+3} = L\pi S(L\pi S(y \oplus \sigma M^t) \oplus M^{t+1}) \oplus M^{t+2} \end{cases} \tag{1}$$

By introducing a nonzero differential $\Delta(M^t, M^{t+1}, M^{t+2}) \neq 0$ the attacker wants to maximise the probability $p = Pr\{\Delta(A1, A2, A3)^{t+3} = 0\}$. That probability may be smaller or larger depending on the permutation σ. Therefore, we need to search for strong σ candidates for the versions SMAC-{1, 3/4, 1/2}, as this will be described in more detail in the next sections. For example, in our simulations we found that for SMAC-1 the strongest found permutation σ_1 ensures the minimum number of active S-boxes is 22 for the case $\Delta t = 3$ (an exampled binary trail for this case is given in Appendix B).

3.2 Searching for a strong σ for SMAC-1 and SMAC-3/4

From the previous subsection it became evident that the minimum number of active S-boxes highly depends on the exact permutation σ. Therefore, one of the goals in this work was to find strong permutation candidates that offer as high level of security as possible for the considered construct.

SMAC-1 vs. -3/4. Searching for a strong permutation for the instance SMAC-3/4 is similar to searching for SMAC-1. We simply insert artificial message blocks with a fixed value $\mathbf{1}^\star$, representing the dummy clocks that happen in SMAC-3/4. Thus, the analysis for SMAC-1 and SMAC-3/4 can be carried out in the same way but with the additional constraint that every 4^{th} differential for SMAC-3/4 is zero, i.e. $\Delta M^t = 0$ for a dummy clock. Note, depending on Δt there can be distinct cases where those dummy clocks occur.

There are $16! \approx 2^{44}$ possible permutations and in this work we aim at performing cryptanalysis on all 2^{44} instances of SMAC, each having a distinct permutation, to extract the strongest candidates. Testing 2^{44} permutations is a challenging task but we were able to narrow it down using several steps of fast filtering and pattern matching[1].

First round of filtering. We have written highly optimised filters for two time frames $\Delta t = 3$ and $\Delta t = 4$ in C/C++. Given a permutation candidate σ, these filters test whether there exists a byte-trail with less number of minimum S-boxes than some preselected threshold. If the number of active S-boxes is below the threshold, the candidate is removed. After the first round of filtering, we obtained the following results:

- SMAC-1. Filters for $\Delta t = 3$ and $\Delta t = 4$ with the threshold at least 20 active S-boxes in an optimal differential trail resulted in 73073 remaining candidates out of 2^{44}.

[1]The feasibility of these simulations was ensured by leveraging two data centres, LUNARC at Lund University and E2C at Ericsson, allowing for parallel testing of these permutation candidates.

- SMAC-3/4. Only one filter was applied in the first round – the case $\Delta t = 3$ with at least 26 active S-boxes, resulted in 1.6 billion candidates.

Second round of filtering. With a larger time frame $\Delta t > 4$ it became more difficult to code specialised filtering functions. We instead used a CP-SAT optimisation solver from OR-Tools [PD] to perform cryptanalysis of the remaining candidates in larger time frames up to $\Delta t = 9$. A generic tool was constructed by utilising constraint programming models to test one or a set of permutation candidates, searching for a differential byte-trail with the minimum number of active S-boxes. The generic tool supports different parameters such as the time frame Δt, position(s) of the dummy message block(s), threshold for the minimum number of S-boxes, et cetera. This tool can also be used to verify and/or test permutations and experiment with SMAC rates other than $\{1, 3/4, 1/2\}$[2].

Patterns. An additional technique to filter out candidates is to collect patterns of weak permutations. If we test a concrete permutation and a filter finds a byte-trail that has less active S-boxes than the desired threshold, we can further examine the permutation indices to spot which indices are contributing to that weak behaviour. It turns out that in many cases there is a particular set of indices that give rise to the low number of active S-boxes, and those permutation indices form a group of candidates that instantly can be skipped during further filtering. By saving such groups as patterns we can significantly reduce the number of permutations we need to test. We collected ∼50 million of unique patterns [3] that helped to significantly truncate the search space.

Simulation results. As the result, we found 20 permutations candidates for SMAC-1 and 40 for SMAC-3/4 (the complete list is given in Appendix A), analysed in the time frame $\Delta t \leq 9$. For SMAC-1, there are only two permutations out of 2^{44} that have at least 20 active S-boxes, and for SMAC-3/4 the found candidates ensure at least 24 active S-boxes.

3.3 Clustering effects

A differential trail typically contains an input difference, output difference, and differences of intermediate variables. Usually, only the input and output differences are known to an attacker. In this case, trails that have the same input and output differences but distinct intermediate differences can cluster together to form a differential with a higher probability, which is called *clustering effect*. Several previous works [BdSF+22, SII24, LPS21] have shown the power of this effect on various ciphers. Therefore, we need to study the clustering effect for the permutation candidates obtained in the previous step of filtering. We have again utilised the CP-SAT solver [PD] from OR-Tools for this part of analysis.

Adaptation of byte-trails for analysis of bit-level clusters. For SMAC, an attacker can prepare a new message that differs from another known message by ΔM, and hope the internal state of SMAC would be the same after a number of clocks, thus resulting in the same MAC value. All intermediate differential bit-trails that have the same ΔM are in the same cluster. However, testing all possible clusters and identifying the strongest one is difficult, since there is an exponential number of concrete input bit-differentials ΔM, and for each ΔM the analysis of its cluster requires enumeration of all intermediate bit-trails. To make this analysis feasible, we study the clustering effect on the level of byte-trails, instead. In particular, we aim to find upper bounds on probabilities of clusters corresponding to only those message differentials that contain optimal byte-trails with the minimum number of active S-boxes. As the result, these bounds apply to all bit-trails that follow the same byte-trails.

[2] The tool, along with a reference implementation and test vectors, is available on GitHub: https://github.com/ONG/smac-tools

[3] As a side note, merging millions of patterns was not a trivial task; thus, we have developed optimised algorithms for handling patterns such as collecting, sorting, merging, and checking for uniqueness, in time $O(N \log N)$.

Analysis of a single byte-level cluster. A single cluster C on the byte level is identified by a concrete fixed byte-differential $\mu(\Delta M)$. Given $\mu(\Delta M)$, we enumerate all intermediate byte-trails which match that certain $\mu(\Delta M)$. Let that set of byte-trails be denoted by S_C. Then the probability of the cluster C is upper bounded by

$$p(C) \leq \sum_{\Psi \in S_C} n_\Psi \cdot max_p(\Psi)$$

where n_Ψ is the number of possible bit-trails matching a certain intermediate byte-trail $\Psi \in S_C$, and $max_p(\Psi)$ is the maximum probability of a single bit-trail that follows the byte-trail Ψ.

In our analysis, we assume that all byte-trails from S_C can be mapped to the same ΔM on the bit-level as well. Therefore, the resulting cluster C, in the way we construct it and taking into account n_Ψs, cannot be smaller than a corresponding valid bit-level cluster, and thus our bound of $p(C)$ provides a theoretical upper bound for the forgery success probability. This also means that in reality the forgery success probability on SMAC is not greater, but likely smaller than what we have derived.

Method to compute n_Ψ. Let us for the moment assume that, given a byte-trail Ψ, its corresponding byte-differential $\mu(\Delta M)$ is assigned with some (unknown to us) fixed bit-level difference ΔM. We propose a simple method to determine whether all other intermediate differential bytes of the byte-trail Ψ can be uniquely derived *on the bit-level*, given a hypothetical fixed bit-differential value of ΔM. Recall that in each round the differential propagates as

$$\begin{cases} \Delta A1^{t+1} = \sigma(\Delta A2^t \oplus \Delta A3^t \oplus \Delta M^t) \\ \Delta A2^{t+1} = L\pi S(\Delta A1^t) \oplus \Delta M^t \\ \Delta A3^{t+1} = L\pi S(\Delta A2^t) \oplus \Delta M^t \\ \Delta A1^{t+2} = \sigma(\Delta A2^{t+1} \oplus \Delta A3^{t+1} \oplus \Delta M^{t+1}) \end{cases} \tag{2}$$

To simplify the analysis, the differential distribution table (DDT) of Rijndael S-box is ignored, and we only consider whether involved S-boxes are active or not. Furthermore, we assume that $\Delta A1^t$, $\Delta A2^t$, and $\Delta A3^t$ are known, and thus $\Delta A1^{t+1}$ is uniquely determined by ΔM on the bit-level due to Eq. (2). Following the given byte-trail Ψ, some bytes can only take the value 0, and for non-zero bytes the following rules are applied repeatedly to determine (most of) the remaining bytes of Ψ on the *bit-level* given Eq. (2), where $\Delta A2_i^{t+1}$ means the i-th byte of $\Delta A2^{t+1}$ and so on for other values.

1. If $\exists i : \Delta A2_i^{t+1}$ is known, then $L\pi S(\Delta A1^t)_i = \Delta A2_i^{t+1} \oplus \Delta M_i^t$.
2. If $\exists i : \Delta A3_i^{t+1}$ is known, then $L\pi S(\Delta A2^t)_i = \Delta A3_i^{t+1} \oplus \Delta M_i^t$.
3. If $\exists i : \sigma^{-1}(\Delta A1^{t+2})_i = 0$ and $\Delta A2_i^{t+1}$ are known, then $\Delta A3_i^{t+1} = \Delta A2_i^{t+1} \oplus \Delta M_i^{t+1}$.
4. If $\exists i : \sigma^{-1}(\Delta A1^{t+2})_i = 0$ and $\Delta A3_i^{t+1}$ are known, then $\Delta A2_i^{t+1} = \Delta A3_i^{t+1} \oplus \Delta M_i^{t+1}$.
5. If the number of unknown byte values in $S(\Delta A1^t)$ (resp. $S(\Delta A2^t)$) is less than or equal to the number of known byte values in $L\pi S(\Delta A1^t)$ (resp. $L\pi S(\Delta A2^t)$), then $S(\Delta A1^t)$ and $L\pi S(\Delta A1^t)$ (resp. $S(\Delta A2^t)$ and $L\pi S(\Delta A2^t)$) are all determined.

Note that $\Delta A1^0 = \Delta A2^0 = \Delta A3^0 = 0$. This way, this method propagates the differential knowledge to intermediate bytes of Ψ round by round, and stops when no new differential bytes can be determined *on the bit-level*. Although the above rules are described on the bit-level when a hypothetical ΔM is known, in our simulations we perform the same steps but on the byte-level where each variable is a binary known/unknown flag.

It is surprising that for almost every Ψ we have tested all intermediate bytes can be uniquely determined after applying these rules, except for a few cases. I.e., given a byte-trail Ψ, there is in most cases only one corresponding bit-trail, which means $n_\Psi = 1$.

In other cases where this method could not determine all values, we have noticed that the remaining (undetermined) bytes are all in the last round and all output from active S-boxes. By checking the number of free variables in the linear system during the 5-th rule, the number of possible bit-trails can be upper bounded. Due to the DDT of the S-box, the number of possible output differences is 2^7 given the input difference. Hence, every free variable in the linear system can only take up to 2^7 possible values. If there are x free variables, then n_Ψ is set to 2^{7x}.

Simulations and results. In the previous filtering stage we have derived a short list of promising permutations. Due to an extremely high complexity of cluster analysis, we picked 5 candidates for SMAC-1 and 9 for SMAC-3/4, such that they cover distinct characteristics (the vector of minimum number of S-boxes for various Δt-scenarios).

For every SMAC variant, permutation, and attack scenario Δt and $(\Delta t, k)$ (where k defines dummy clocks), we first enumerate all clusters identified by distinct byte-differentials $\mu(\Delta M)$ that include at least one optimal byte-trail with the minimum number of active S-boxes. Then for each such cluster we compute the upper bound of the forgery attack success probability by using the above method, where the enumeration of intermediate byte-trails as well as computation of $max_p(\Psi)$ was done with OR-Tools. In the end, we get the maximum probability over all attack scenarios for each SMAC variant and permutation, from where the most secure permutations are determined.

For SMAC-1, we found that σ_1, σ_{11}, and σ_{17} are the three strongest candidates with similar forgery probabilities upper bounded by $2^{-118.95}$, $2^{-119.41}$, and $2^{-118.40}$, respectively. Our preference goes to σ_1 for the reason that it is one of only two permutations out of 2^{44} that has at least 20 active S-boxes in all Δt scenarios, while σ_{11} and σ_{17} ensures only 19 active S-boxes; also, both σ_{11} and σ_{17} are weaker than σ_1 in respect to other analyses[4].

For SMAC-3/4, we found that σ_{37} and σ_{42} are the strongest candidates with similar forgery probabilities upper bounded by $2^{-151.21}$ and $2^{-152.29}$, respectively. However, in this case, we would prefer σ_{42} with a slightly better security.

As the number of active S-boxes grows rapidly with larger Δt, we believe that the existence of a forgery differential attack with complexity much lower than the claimed security level and time frame $\Delta t > 9$ is not likely. More detailed results of the cluster analysis on permutation candidates can be found in Appendix A.

3.4 Selection of σ for SMAC-1/2

In order to find decent permutation candidates for SMAC-1/2, we collected all permutations that have been considered or filtered out in previous stages of this work (except that large 1.6 billion set collected for SMAC-3/4), and analysed around 180 million permutations[5] resulting in 16 promising candidates that have at least 41 active S-boxes in $\Delta t \leq 9$. Afterwords, we perform a cluster analysis on two of them, σ_{61} and σ_{69} (the full list of candidates can be found in Appendix A).

For SMAC-1/2, we adopt a rough method to estimate a cluster probability such as, when considering a single byte-trail we force the CP solver to maximise the number n_6 of active S-boxes with the optimal probability 2^{-6}. Upon reaching a timeout the solver returns with the range of n_6, from where we pick the maximum bound and assume all other active S-boxes have probability 2^{-7}, this way we upper bound the probability of a particular trail. Note that if there are free variables in the trail then we should also include these into the probability, same as we did in SMAC-{1, 3/4}. The estimates we

[4] For example, in GnD test G5 (Table 3) both σ_{11} and σ_{17} are not secure in all $d < 9$.

[5] It was, however, infeasible for us to test all 2^{44} permutations in this rate, and a stronger candidate might exist.

receive for all trails are used to derive an upper bound for all clusters and therefore the forgery success probability.

The two analysed permutations σ_{61} and σ_{69} have the forgery probabilities upper bounded by $2^{-252.30}$ and $2^{-256.00}$, respectively. However, σ_{69} was identified to have some weaknesses in respect to other analyses[6], and for this reason we have selected σ_{61} for the half-rate version.

3.5 Justification for the constant $1^\star$

The `MixColumn` linear transformation has a specific property such as if the input bytes are all equal, $X = \{x\}^{16}$, then the result $Y = \texttt{MixColumn}(X)$ preserves the same property and $Y = \{y\}^{16}$. All other operations, such as XOR, σ, `ShiftRows`, and `SubBytes`, also preserve this property. I.e., if all three registers $A1, A2, A3$ have that property in some certain time, then that property preserves over rounds *if* the message block $M = 0$, which may, in particular, affect the randomness of the initialisation and finalisation phases. In the initialisation phase, this property can further generate a weak key class. In order to remove this property, we add $\mathbf{1}^\star$ as the round key to the state during the `InitFinal` function, as well as for dummy clocks in SMAC-{3/4, 1/2}.

3.6 Arguments behind the PRP-PRF switch

The ending XOR with the input in the function `InitFinal` converts it from a pseudo-random permutation (PRP) to a non-invertible pseudo-random function (PRF), similarly to the FP(1) mode of operation in stream ciphers [HK15]. This protects both the secret key and the state sequence. For example, suppose the state in some time instance is recovered, say, through a side-channel attack. In that case, it is not possible to revert the state back to the start of the initialisation phase and recover the secret key, the highest asset to be protected. Moreover, since t bits of the internal state become the final MAC tag value, it also makes sense to protect the final state.

The `InitFinal` procedure can be simplified as

$$Y = \Pi^d(X) \oplus X,$$

where X, Y are 384-bit variables and Π is the SMAC round function with $\mathbf{1}^\star$as the message. The ending $\oplus X$ converts the PRP Π into a PRF. This is a standard technique and is used in many designs, for example, in MILENAGE for computing OP_c from OP [3GP], or in Grøstl [GKM+09] for the output transformation. The theoretical security of the finalisation function may be derived from e.g. the security proof of Davies-Meyer construct where $g(k, m) = E_k(m) \oplus m$ is proved to be a collision-resistant one-way function, given that E_k is an ideal block cipher [BRS02] and the same applies when the key k is fixed.

3.7 Internal state size and TMTO attacks

Assume that an attacker can observe the full 384-bit output Y (and not just the tag that is of maximum size 256 bits). What is the complexity of reverting Y into X? In case of PRP that would be a 1-to-1 mapping and the reverting algorithm is trivial – just clock backwards d times. However, in case of a PRF that mapping would in most cases have between 0 to 2 solutions and it is not trivial how to revert it as $\oplus X$ may be viewed as a masking of $\Pi(X)$. To revert that PRF, one may try a TMTO trade-off attack of complexity $T = M = D = O(2^{192})$ by just building a table of $M = O(2^{192})$ (X, Y) pairs and then ask for $D = O(2^{192})$ different Y's. This is a state-recovery attack. The full Y is not available

[6] For example, in GnD test G5 (Table 3) σ_{69} is only partially secure in $d = 7$ (only 9 bytes of 48 are secure) and $d = 8$ (36/48).

from the SMAC tag, but if the nonce is misused in the verification oracle one can combine a few accepted tags from the same IV and fixed messages to form an output Y unique for each X (e.g. see Section 3.13). If the tag size is 32 bits then 2^{40} calls to the verification oracle are sufficient to get on average 2^8 accepted tags.

Generic key-recovery TMTO attacks are similarly also valid for our construct. For example, one can create a large table that maps a subset of the key and IV space to MAC tags for a predefined set of messages, and when the attacker observes tags also found in the table the full key is recovered. This TMTO attack would have a complexity around $T = M \approx O(2^{192})$ with data $D = O(2^{192})$ tags generated from different keys and IV pairs. Better is to fix the IV and have the same TMTO attack on the key only, requiring $T = M = D \approx O(2^{128})$. Allowing a large precomputation cost, one can reduce the memory and data cost by Hellman's approach [Hel80] and building Rainbow tables. It does not, however, offer better performance than the generic case of a search for a 256-bit key.

The state size 3×128 bits ensures a high enough resistance against internal state collision attacks in birthday paradox and TMTO settings. A single-state collision may happen naturally among 2^{192} collected pseudo-random states.

3.8 Avalanche effect on full registers

A brief analysis of the initialisation/finalisation phases can also be given by the avalanche effect on the level of registers depending on the number of clocks d. The results are given in Table 2 where, for the sake of notation, by k^x we denote that the initial value of Ak has been involved x times in an expression for the resulting register after d clocks. As the result, after $d = 3$ clocks the register $A1$ already involves all three initial values, and after $d = 5$ clocks each of the three registers involves the whole initial state.

Table 2: The avalanche effect on registers depending on the number of clocks.

clocks, d	0	1	2	3	4	5	6	7	8	9
$A1$	1^1	$2^1 3^1$	$1^1 2^1$	$1^1 2^1 3^1$	$1^1 2^2 3^1$	$1^2 2^2 3^1$	$1^2 2^3 3^2$	$1^3 2^4 3^2$	$1^4 2^5 3^3$	$1^5 2^7 3^4$
$A2$	2^1	1^1	$2^1 3^1$	$1^1 2^1$	$1^1 2^1 3^1$	$1^1 2^2 3^1$	$1^2 2^2 3^1$	$1^2 2^3 3^2$	$1^3 2^4 3^2$	$1^4 2^5 3^3$
$A3$	3^1	2^1	1^1	$2^1 3^1$	$1^1 2^1$	$1^1 2^1 3^1$	$1^1 2^2 3^1$	$1^2 2^2 3^1$	$1^2 2^3 3^2$	$1^3 2^4 3^2$

3.9 Guess and determine attacks

In this section, we consider the `InitFinal` function

$$(A1', A2', A3') = \Pi^d(A1, A2, A3, \mathbf{1}^\star) \oplus (A1, A2, A3),$$

and study the complexity of a generic guess-and-determine attack for various scenarios where some of the input/output register values are known, and we want to derive the remaining values through guessing the smallest number of other unknown bytes. We will model relations on the byte level, and while all operations are simple, it is only `MixColumn` that is more complex to model which includes 56 relations per a single 4-to-4 byte `MixColumn`. In order to find a (almost) smallest guess base for our GnD attack scenarios, we utilise the tool Autoguess from [HE22], as well as our own tool developed solely for this project (a brief description can be found in Appendix D).

We have three sets of GnD scenarios. First of all, in G1 we would like to understand how good that PRF function is, i.e., given the complete output, how many bytes need to be guessed to revert that PRF back to the input. In G2 we consider the case when two input states are related through, e.g., a known differential, and both output states are fully available to the attacker. We see that the additional knowledge of an extra output

state in G2 does not much help in a GnD attack as the complexity to recover the initial state is similar to G1.

The next set of scenarios G3-5 addresses the security of the initialisation phase where, as a hypothetical assumption, we let the whole state after initialisation to be known to the attacker, as well as some values of the input registers. In these scenarios we are interested in the minimum guess base to recover the missing input register (or even a single byte[7]), where the secret key may actually be settled. These attack vectors may become realistic if e.g. one device performs the initialisation and bypasses the computed output state to the second device for actual MACing, but that second device may be compromised.

Table 3: The observed minimum sizes of guess bases for σ_1, σ_{42}, and σ_{61} under various scenarios received from heuristic tools, in terms of the number of bytes to be guessed.

Scen-	Trivial	Known input/output	Number of rounds, d									
ario	guess	registers and bytes	1	2	3	4	5	6	7	8	9	10
General: how strong the stand-alone PRF function is.												
G1	48	All output regs. $A1', A2', A3'$	8	16	16	26	31	32	40	44	48	48
General: state recovery for two related input states given corresponding output states.												
G2	48	All 2×48 output bytes	8	16	16	26	31	32	40	46	48	48
Initialisation: key recovery (or even a single byte) given the complete state after initialisation and some values at loading time.												
G3	32	One of $A1, A2, A3$ and $A1', A2', A3'$	0	0	0	10	15	16	24	28	32	32
G4	16	Two of $A1, A2, A3$ and $A1', A2', A3'$	0	0	0	0	0	0	8	13	16	16
G5	1	Any 47 input bytes and $A1', A2', A3'$	0	0	0	0	0	0	1	1	1	1
Finalisation: state recovery given a tag taken from various registers.												
G6	48	One of output regs. $A1', A2', A3'$	32	32	32	32	32	32	40	44	48	48
G7	48	Two of output regs. $A1', A2', A3'$	16	16	16	26	31	32	40	44	48	48
G8	48	20 output bytes $A1', A2'_{[0..3]}$	28	28	28	30	32	34	45	48	48	48
G9	48	20 output bytes $A2', A3'_{[0..3]}$	28	28	28	29	32	32	45	48	48	48

In the third set of scenarios G6-9, we analyse the finalisation part where, given the knowledge of one or more output bytes (e.g. through the MAC tag), we wonder about the complexity to recover the internal state before the finalisation phase.

The absolute security level for all these scenarios is that guessing at most 1/16/32/48 bytes of the unknown input registers is enough to recover all other variables – we call it as a *trivial guess*.

All GnD scenarios and the smallest size of the guess base that we managed to derive and observe by using heuristic tools are given in Table 3. Since these tools are heuristic, a smaller guess base may still exist. However, the results that we received are still good indications on what the size of the guess base can be, and how it grows with the number of rounds d.

Notably, the results of these simulations demonstrate that with $d = 9$ we seem getting the absolute maximum possible security level in all GnD scenarios, although many of them are only theoretical. To note, the highest security level of 384 bits is not really needed as it is already much larger than it could be required for a possible SMAC use case, thus the number of rounds d could actually be lower for certain applications, and may also be different for the initialisation and finalisation phases.

[7] G5 simulates a scenario when all key bytes except one are guessed and the complete output is also known; it demonstrates that with $d \geq 7$ clocks that single byte is still an unknown variable and cannot be determined through all other 95 known bytes.

3.10 MDM and cube tests

In this section, we perform the MDM test and cube attack on `InitFinal` to check how many rounds are needed to fully mix the input bits. The initial state in time $t = 0$ is supposed to be pseudo-random by the initial d clocks, which shuffles the input parameters and the secret key. The first preimage resistance should be ensured by the ending d clocks, which makes it hard to find a message that results in a particular hash value. Note that practical distinguishers based on these two methods require fixing some input bits to known values and enumerating another subset of input bits which is called a *cube*. Meanwhile, both methods have to compute the summation of the outputs. However, these requirements cannot be satisfied simultaneously for either the initialisation or finalisation phase. Because the output of the initialisation phase and the input of the finalisation phase are secret to attackers. To analyse both phases, we view them as the same stand-alone function, `InitFinal`, and assume that all the inputs and outputs can be obtained.

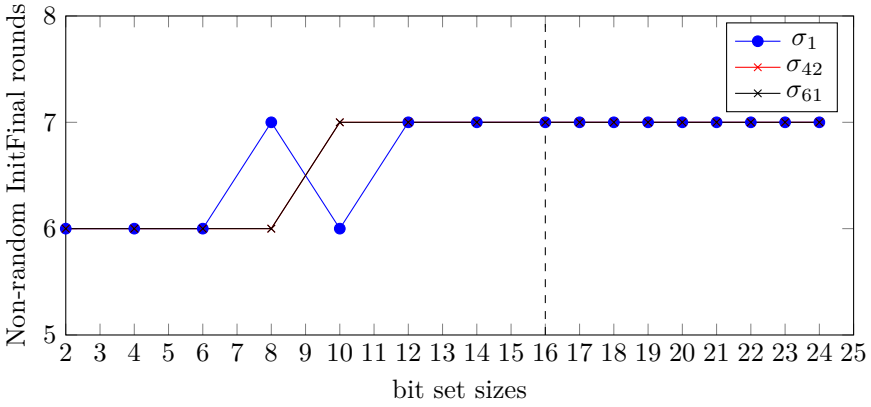

Figure 3: MDM tests of `InitFinal`.

In MDM test for a Boolean function, one fixes values to the input bits outside the cube, enumerating all possible values of the bits in the cube, and then sums the output values of the function. For a random Boolean function, the result will be 1 with probability 1/2 while there is a bias for a non-random one. [Sta10] provides a greedy algorithm to find a good cube that detects non-randomness through ciphers. We regard the 384 output bits of `InitFinal` as 384 Boolean functions and take this algorithm to check their non-randomness. Our test starts with the worst 2-bit set that shows the longest non-random rounds. In each step, we add two new bits that give the worst randomness. When the size of the bit set reaches 16, the time complexity of finding the next two bits is too high, so we have to switch to adding one new bit in the next steps until the bit set has 24 bits. To test with a larger bit set, a more powerful computer is needed. Our results are shown in Figure 3. It can be seen that the first 7 rounds fail the MDM test, ensuring a good mixing effect.

Table 4: Cube attacks on reduced-round of the initialisation phase (the results are the same for σ_1, σ_{42}, and σ_{61}).

Rounds, d	3	4	5	6	7	8	9
cube size $\|I\|$	7	7	7	103	103	128	128
degree d	21	126	231	231	255	254	255
involved key size $\|J\|$	24	152	256	256	256	256	256
time complexity	$2^{30.99}$	2^{159}	$> 2^{256}$	$> 2^{256}$	$> 2^{256}$	$> 2^{256}$	$> 2^{256}$

In cube attack, by computing the summation of the outputs, attackers aim to recover the

superpoly of the chosen cube. After the division property was proposed by Todo [Tod15], it was further used in [TIHM17] to find the set of key bits J that are involved in the superpoly of the given cube I. This method was improved by Wang *et al.* [WHT+18], which reduces the time complexity of the attack to $2^{|I|} \cdot \sum_{i=0}^{d} \binom{|J|}{i}$ where d is the degree of the superpoly. We evaluate the security of `InitFinal` against cube attacks by using the method described in these papers. Our model is a MILP model. The linear operations can be described by the models of XOR and COPY from [XZBL16] while the model of the Rijndael S-box is given by [Tod15]. Several different cubes were tested and Table 4 shows the best results found by our model. One can see that, after 8 rounds, the degree is almost full which matches the result in Table 6. Meanwhile, all key bits are involved in this superpoly and the time complexity of the cube attack is larger than 2^{256}.

3.11 Differential attacks on initialisation

Another test that we performed is a differential analysis where the initial state after loading $(A1, A2, A3)$ may have a difference $\Delta(A1, A2, A3)$, then we check the minimum number of active S-boxes after d initialisation rounds that bring the state to any other difference $\Delta(A1', A2', A3')$ (which may be zero or nonzero). One may argue that since IV and IV′ for the pair of messages can be selected or even be fixed to certain values, it might help the Key-differential to propagate through the initialisation phase more efficiently. However, since the IV is loaded into $A3$, we see that the very first clock would compute $A3 \oplus A2$ where $A2$ is the lower part of the secret key, thus making the intermediate result unknown. This way, considering a differential over both IV and Key parts would be a generic differential attack on the initialisation phase. This observation motivates to reserve the lower part of the key for $A2$, then if the original key is larger than 128 bits, the remaining bits to be placed into $A1$.

Table 5: Minimum number of active S-boxes in a differential trail through `InitFinal`.

Rounds, d	3	4	5	6	7	8	9	10
#S-boxes for $\sigma_1/\sigma_{42}/\sigma_{61}$	5	6	13	30	32	40	56/49/54	70/64/69

In our simulations, we find optimal trails that activate the minimum number of S-boxes in the initialisation phase for each value of d and the results are given in Table 5. As an example, for $d = 6$ any differential trail involves at least 30 active S-boxes that makes the trail probability upper bounded by 2^{-180}, which is far smaller than the target 2^{-160} in SMAC-3/4.

3.12 Output MAC tag registers

Finally, in order to determine which registers should serve as the source of the final tag, we performed yet another MILP-aided analysis which resulted in the degree bounds of the Boolean functions of the registers' bits. This method comes from [WHT+18] where the authors used it to determine degrees of superpolies. We emphasise that the maximum degree is 383. This is because, without the PRP-PRF switch, `InitFinal` is a permutation whose degree is upper bounded by 383 and the switch does not change the final degree.

Table 6: Degree bounds of Boolean functions of the registers.

Rounds, d	3	4	5	6	7	8	9	10
A1	28	49	196	280	[352, 356]	376	382	383
A2	49	133	280	[352, 356]	376	382	383	383
A3	28	196	232	[352, 356]	372	382	383	383

From the results given in Table 6, we see that the bound of the register $A1$ is always slightly behind the other two registers $A2$ and $A3$, for different ds. An obvious reason is that $A1$ is the linear combination of the previous $A2$ and $A3$ which does not increase the degree. This motivates us to produce the output tag from $A2$ first, and if more bits are needed, in SMAC-{3/4, 1/2} with $\tau > 128$, then we take them from $A3$.

3.13 State recovery attack with nonce-misuse queries

Atomic step. As a simplified scenario, let us demonstrate feasibility of a state recovery using nonce-misuse queries when (K, IV) is fixed. Recall a differential forgery success probability as discussed in Sections 3.1 and 3.3. Let us take the case SMAC-1, $\Delta t = 3$, and assume we get two messages M and $M' = M \oplus \Delta M$ where the message difference ΔM only happens during the time width $[t..t+2]$ and results in $\Delta(A1, A2, A3)^{t+3} = 0$ after these 3 clocks. In this case we have a state collision. Now recall the middle equation from Eq. (1) which is $A2^{t+3} = L\pi S(x \oplus \sigma M^{t+1}) \oplus M^{t+2}$, where x, y, z are one-to-one substitutions from $(A1, A2, A3)^t$. Since $\Delta A2^{t+3} = 0$, and both M and M' are known to us, we derive:

$$\begin{aligned} L\pi S(x \oplus \sigma M^{t+1}) \oplus M^{t+2} &= L\pi S(x \oplus \sigma(M^{t+1} \oplus \Delta M^{t+1})) \oplus (M^{t+2} \oplus \Delta M^{t+2}) \\ \Rightarrow L\pi S(x') &= L\pi S(x' \oplus \sigma \Delta M^{t+1}) \oplus \Delta M^{t+2}, \text{ where } x' = x \oplus \sigma M^{t+1} \\ \Rightarrow S(x') &= S(x' \oplus a) \oplus b, \text{ where } a = \sigma \Delta M^{t+1}, b = (L\pi)^{-1} \Delta M^{t+2} \end{aligned}$$

Due to the state collision, for any $i \in [0..15]$ we get either $a_i = b_i = 0$ or $a_i \neq 0 \wedge b_i \neq 0$, and these byte values are derived from the (M, M') pair and thus known. For the latter case where a_i and b_i are both nonzero, the x'_i may have only 2 (in most cases) or 4 possible values. In this way we learn about the internal state, represented by the substitution triple (x, y, z). We can then take the other two equations from Eq. (1) and analyse these in a similar way to learn about the unknown y, z. The idea of using a differential trail for the state recovery is not new, see e.g. [HII+22], and here we extend that idea on SMAC-1.

Repeat the atomic step several times. In SMAC-1, the optimal byte trail involves at least 20 active S-boxes, which means that from a single atomic step we learn around 20 bytes of the internal state. We can repeat the atomic step by using a different byte trail with a different $\mu(\Delta M)$ (for the same time instance t in all repeated atomic steps, but may involve different Δt and distinct trails), and recover new bytes of the internal state. After 3-4 such atomic steps, the complete 48-byte internal state can be derived.

State collision detection. To detect a true state collision, and not a random tag collision, we can append a single block to both M and M' and query the oracle whether the sequences $(M||1), (M||2) \ldots$ and $(M'||1), (M'||2) \ldots$ still produce the same tag. In case of a random tag collision, this will not be the case.

Complexity to get a related pair (M, M'). Assume for the moment that we can make queries to the sender oracle (although this scenario is not realistic since in that case the attacker already has access to the universal oracle). In a naïve approach, we choose ΔM, pick a random M and derive $M' = M \oplus \Delta M$, then call the oracle to get the sequence of tags, and thereby determine whether it is a state collision or not. If the success probability of the state collision for the chosen ΔM is 2^{-s}, then we need to make $O(2^s)$ queries.

However, that complexity could be improved as follows. We pick a byte-differential $\mu(\Delta M)$, then make around $O(N = \sqrt{2^s})$ queries to the oracle with different messages $M_1, \ldots, M_N$, and get relevant tag-sequences $T_1, \ldots, T_N$ (each sequence of tags should cumulatively have size at least s bits) for each message. Each message is constructed to follow the chosen byte-differential $\mu(\Delta M)$ such that if for any byte index i we have $\mu(\Delta M)[i] = 0$ then in every message that byte $M_k[i]$ is a constant value for all $k \in [1..N]$ (we can choose that constant byte at random); and when $\mu(\Delta M)[i] = 1$ then $M_k[i]$ is

picked at random in every message independently. In the end, we would get $2^{s/2}$ pairs $\{k \in [1..N] : (M_k, T_k)\}$, and due to the birthday paradox there should be a pair (M_a, T_a) and (M_b, T_b) in the list such that $T_a = T_b$, meaning that we fall into the state collision. The actual bit-differential is $\Delta M = M_a \oplus M_b$, and the probability that this concrete bit-differential value follows the chosen byte-differential $\mu(\Delta M)$ is high (we skip further details at this point).

In order to find a related pair of messages among N collected, we can sort the list after the tag-sequences in time $O(N \log N)$, then find a matching $T_a = T_b$ in linear time. The sorting complexity can be decreased by also using hash tables, thus we should conservatively regard this step as the minimum $O(2^{s/2})$.

The overall complexity of the state recovery attack by using nonce-misuse queries to the sender oracle is at least $O(2^{s/2})$ queries, or more. If the attacker can only use a verification oracle, then the complexity is at least $O(2^{s/2+\tau})$. For SMAC-{1, 3/4, 1/2}, the minimum values for s are $\{118, 152, 252\}$, respectively, thus the absolute lower bounds for this attack is at least $O(2^{\{59,76,126\}+\tau})$ queries. These estimates are highly conservative.

Note also that this attack does not lead to a key recovery, and the universal forgery can create messages only for a certain pair of (K, IV). Also, the above "birthday paradox" improvement may not work if the space of valid bit-differentials ΔM that follow the selected byte-differential $\mu(\Delta M)$ contains additional constraints on the bit-trails (e.g., not all ΔM are possible for the state collision to happen), thus the attack complexity may actually be much higher, up to $O(2^{s+\tau})$ queries. We leave this study as an open question to refine the nonce-misuse attack complexity in the future.

4 Aggregated mode version SMAC-1×n

The SMAC compression function can be used in an *aggregated mode* of operation, where multiple cores of SMAC compress the stream of message blocks in parallel, then their end results are combined to produce a single output MAC tag. The mode of aggregation can be compared to a parallelised implementation of GHASH.

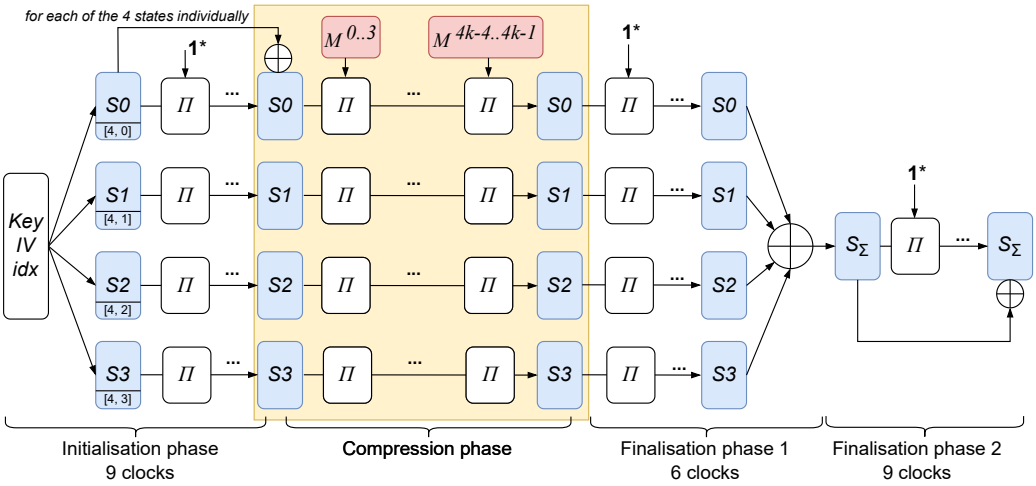

Figure 4: Exampled construct of SMAC-1×4 with 4 parallel compressions.

In this section we present the aggregated mode of SMAC-1, called SMAC-1×n, where n is the level of aggregation, or, by other words, the number of parallel streams. Note that parallel streams can be implemented efficiently in software by e.g. utilising wider SIMD

Algorithm 4 SMAC-1×n

function SMAC-1×$n(n, \tau, K0||K1, IV, \mathcal{A}, \mathcal{C}) \to Tag$
  Construct 16-byte block $L = \texttt{LittleEndian64}(len(\mathcal{A}))||\texttt{LittleEndian64}(len(\mathcal{C}))$.
  Pad $\mathcal{A}$ and $\mathcal{C}$ with zeroes to align with full 16-byte blocks.
  Concatenate $\mathcal{M} = (\mathcal{A}||\mathcal{C}||L)$ and pad with zeroes to align with $16n$-byte blocks.
  Create n states $S_0, \ldots, S_{n-1}$ each having three registers $S_k.A1$, $S_k.A2$, $S_k.A3$
  Patch the IV as $IV[15] = (n-1) \cdot 16$.
  Initialise $S_k.A1 = K1$, $S_k.A2 = K0$, $S_k.A3 = IV \oplus (k \cdot 2^{120})$, $\forall k \in [0..n-1]$.
  for $i = 1..9$ **do** ▷ Initialisation loop
    $\forall k \in [0..n-1] : \ (S_k.A1, S_k.A2, S_k.A3) = \Pi(S_k.A1, S_k.A2, S_k.A3, \mathbf{1}^\star)$
  $\forall k \in [0..n-1] : \ (S_k.A1, S_k.A2, S_k.A3) = (S_k.A1, S_k.A2, S_k.A3) \oplus (K1, K0, IV)$
  Divide $\mathcal{M}$ into mn sub-blocks M^i of size 16 bytes each ▷ Compression
  for $i = 0..m-1$ **do**
    $\forall k \in [0..n-1] : \ (S_k.A1, S_k.A2, S_k.A3) = \Pi(S_k.A1, S_k.A2, S_k.A3, M^{i \cdot n+k})$
  for $i = 1..6$ **do** ▷ Finalisation phase 1
    $\forall k \in [0..n-1] : \ (S_k.A1, S_k.A2, S_k.A3) = \Pi(S_k.A1, S_k.A2, S_k.A3, \mathbf{1}^\star)$
  $(A1, A2, A3) = \bigoplus_{k \in [0..n-1]} (S_k.A1, S_k.A2, S_k.A3)$
  $A2' = A2$ ▷ Finalisation phase 2
  for $i = 1..9$ **do**
    $(A1, A2, A3) = \Pi(A1, A2, A3, \mathbf{1}^\star)$
  $Tag = (A2 \oplus A2')_\tau$

registers such as 512-bit ZMM0..ZMM31, and modern instructions such as AVX-512, thus making an aggregated mode as an attractive option for various cryptographic primitives.

As a concrete example, we present the design of SMAC-1×4 with four parallel compression streams. The value of user-defined IV is limited to 15 bytes $IV[0..14]$, while the last byte of $IV[15]$ is reserved by the scheme to encode the total number of streams (4 in this case) and a corresponding stream index (from 0 to 3, in this case). The message M is constructed by concatenation of zero-padded 16-byte blocks of AAD, ciphertext, and the "lengths" block. The initialisation phase clocks 9 times in four parallel states, and the same padding for every stream state $(K1, K0, IV)$ using the patched IV of the first stream. The finalisation step is, however, modified and split into two sub-phases. In the first phase, all four parallel cores perform 6 dummy clocks. After that all states are bitwise XORed together to form a single state. That single state is then finalised as SMAC-1 with 9 finalisation clocks with the ending PRP-PRF switch. The MAC tag is taken as $A2_\tau$, where $\tau \in [12..128]$. Schematically, SMAC-1 × 4 is depicted in Figure 4.

A more general scheme of SMAC-1×n is defined by Algorithm 4. The number of parallel streams is $n \in [1, 16]$ (i.e., the design allows maximum 16 streams), and a 1-byte encoding is $IV[15] = (n-1) \cdot 16 + i$, for the ith stream, where the stream index is $i \in [0, n-1]$. A similar definition can be obtained for SMAC-{3/4, 1/2}×n.

4.1 Security arguments for the aggregated mode

In Section 3 we studied the security of SMAC-r that compresses a message using a single state. In this section we consider possible security threats related to the use of the aggregated mode. SMAC-$r \times n$ has four phases, each requiring an analysis of possible attack landscapes.

Compression phase. Let the message be split into n (interleaved) chunks $M_0, \ldots, M_{n-1}$, each is fed into corresponding compression stream. The aggregated mode assumes *independent random* initial states $S_0, \ldots, S_{n-1}$ of these n streams. Obviously, each engine works as an independent SMAC-r and its security was studied in previous sections.

Initialisation phase. The source of initialisation for all initial states is a pair (Key, IV). Clearly, $S_0, S_1, \ldots$ must be *(pseudo-)independent* from each other, which is the assumption of the compression phase, and *ordered*. The latter requirement prevents trivial attack vectors where the initial states are related such as $(S'_0, S'_1) = (S_1, S_0)$, allowing for a MAC collision by simply swapping M_0 and M_1. The *order* is achieved by the use of *index*.

Finalisation phase 2. This is a direct replica of the plain SMAC-r finalisation and its security was also analysed earlier.

Finalisation phase 1. In a possible attack scenario, a forgery forms a message difference that is mapped to state differences in n parallel compression streams. The goal is for these state differences to cancel out and become zero after the XOR operation, as illustrated in Figure 5, with a success probability larger than the security level anticipates.

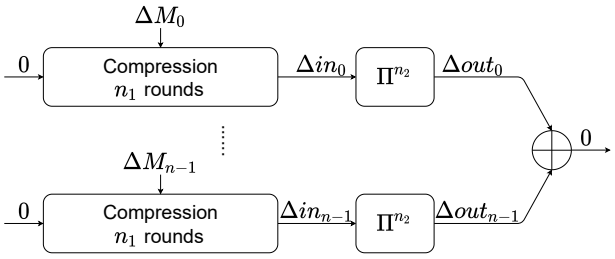

Figure 5: Outline of a differential attack on aggregated mode.

The n message differences $\Delta M_i, i \in [0..n-1]$, introduced in the last n_1 compression rounds, result in the state differences Δin_i in each of the n parallel engines. If any of them is zero, then the analysis case is reduces to a smaller n. When all of them are zero, it is a scenario as in Section 3.3. Thus, we can assume that all Δin_i are non-zero.

As a simplified argument, assume we only have two streams and a forgery can *directly* insert Δin_0 and Δin_1. Based on Table 5, this would trigger at least 2×30 active S-boxes meaning that the success probability of such a forgery attack bounded by around $2^{-6 \cdot 60} = 2^{-360}$, excluding clustering effects.

It is, however, more challenging to analyse multiple streams in general, especially considering clustering effects. Let the finalisation phase 1 contain n_2 dummy clocks ($n_2 = 6$ in our case) and the output state differentials in the corresponding compression streams are $\Delta out_i, i \in [0..n-1]$. Denote the probability of the differential trail over $n_1 + n_2$ rounds by $p(\Delta out_i)$, then the probability of the collision event $\bigoplus_{i=0}^{n-1} \Delta out_i = 0$ after the XOR operation can be derived as

$$p_{collision} = \sum_{\substack{\Delta out_0, \ldots, \Delta out_{n-2}, \\ \Delta out_{n-1} = \bigoplus_{i=0}^{n-2} \Delta out_i}} \prod_{i=0}^{n-1} p(\Delta out_i) \tag{3}$$

Direct computation of Eq. (3) is infeasible since the state of SMAC-r has 384 bits and it seems we have to consider all 2^{384} possibilities. However, following the analysis of Farfalle [BDH+17] and our previous differential analysis of SMAC (especially the analysis of clustering effects), we adopt the following presumption: *The probability $p(\Delta out_i)$ is dominated by those differential characteristics that have the minimum number of active S-boxes.* Let the number of dominating characteristics be upper bounded by c, and $\omega[n_1, n_2]$ be the minimum number of active S-boxes over the final $(n_1 + n_2)$ rounds processing for a single stream. Then under the presumption, we have

$$\prod_{i=0}^{n-1} p(\Delta out_i) \leq \prod_{i=0}^{n-1} \left(c \cdot 2^{-6 \cdot \omega[n_1, n_2]} \right) = c^n \cdot 2^{-6n \cdot \omega[n_1, n_2]}$$

The number of valid Δout_i can be estimated through the number of active S-boxes in the last two rounds since each byte would then pass through an S-box at least once (more details are given in Appendix E). Note that for dominating characteristics, the last two rounds can only have at most $\omega[n_1, n_2] - \omega[n_1, n_2 - 2]$ active S-boxes. Therefore, the number of valid Δout_i can be roughly estimated as $\sum_{k=1}^{\omega[n_1,n_2]-\omega[n_1,n_2-2]} \binom{48}{k} \cdot 2^{8k}$. By combining all the discussed elements, we obtain

$$p_{collision} \lessapprox \left(\sum_{k=1}^{\omega[n_1,n_2]-\omega[n_1,n_2-2]} \binom{48}{k} \cdot 2^{8k} \right)^{n-1} \cdot c^n \cdot 2^{-6n \cdot \omega[n_1,n_2]}$$

In order to find $\omega[n_1, n_2]$ under different n_1 and n_2, we utilise our tool mentioned in Section 3.2. Based on enumerations in Section 3.3 we take $c = 2^{12}$, which results in the collision probability upper bounded by $2^{-299.36}$. This offers a large security margin.

Based on the above analysis, we conclude that the security and limitations of SMAC-$r \times n$ are similar to those of SMAC-r.

5 Software evaluation

The compression core function of SMAC can be implemented using only a few SIMD instructions on modern CPUs, and our assessment is that SMAC is a fast and competitive design in both software and hardware.

```
void SMAC_Compress(__m128i& A1, __m128i& A2, __m128i& A3, __m128i* msg)
{       __m128i M = msg ? *msg : const1;  // if msg=NULL then dummy clock
        __m128i T = Sigma(Xor3(A2, A3, M));
        A3 = AesRound(A2, M);
        A2 = AesRound(A1, M);
        A1 = T;
}
```

Listing 1: SMAC compression function (implementation sketch).

In the first step of performance evaluation we compare only the most critical compression cores for various algorithms, excluding any precomputations, initialisation and finalisation phases, and running on a random data of length 1 MB. The results are given in Table 7. All tested critical cores, except GHASH/AES-GCM, were implemented by us, highly optimised and partially unrolled to gain the maximum performance. Selected set of algorithms for comparison is similar to that used in [BBL+24]. The core of the base version SMAC-1 performs 64% faster than the base version PetitMac, and the version that benefits from a high level of parallelisation SMAC-1 × 16 runs 95% faster than LeMac, up to the record 1 Tbps. All tests were performed on a user-grade laptop with Intel Core i5-1145G72.60/4.40GHz, single threaded, and the performance numbers that we received are similar to those from [BBL+24], which additionally confirms that our measurement techniques and results are correct.

In the second step we evaluate complete SMAC implementations including all initialisation and finalisation steps, padding and aligning, given various lengths of data. The results are given in Table 8. We include measurements for GHASH core implemented in OpenSSL 3.0.0 (in an optimised assembly code) as a competitive reference. For the evaluation of complete algorithms LeMac and PetitMac we took authors' implementations[8] and added minor improvements to speed them up further. The maximum performance that we reached for the base version SMAC-1 is 156 Gbps, and for the aggregated version SMAC-1 × 8 it is 925 Gbps. These numbers are higher compared to those of Petit&LeMac, respectively. Also note that SMAC demonstrated its competitiveness even with short messages.

[8] Implementations of LeMac and PetitMac, provided by the authors of [BBL+24], can be found on https://github.com/AugustinBariant/Implementations_LeMac_PetitMac.

Table 7: Performance evaluation of critical compression cores on 1 MB of data, excluding the initialisation, finalisation, and other routines.

Algorithm	Characteristics, in blocks, per round			Performance		
	State size	AES_R calls	Messages	Gbps	cpb	[BBL+24]
Constraint base designs that process one message per round						
AEGIS-128(AD) [WP14]	5	5	1	93	0.376	0.389
PetitMac [BBL+24]	6	2	1	93	0.376	0.376
SMAC-1 (this paper)	3	2	1	156	0.226	—
Previous notable constructs that tend to utilise parallelisation						
AEGIS-128L(AD) [WP14]	8	8	2	187	0.187	0.195
GHASH/GCM(AD) [NIS]	—	—	—	213	0.165	0.286
Tiaoxin-346v2(AD) [Nik14]	11	6	2	228	0.154	0.121
Rocca-S(AD) [NFI24]	7	6	2	265	0.132	0.151
Rocca(AD) [SLN+21]	8	4	2	274	0.128	0.149
Jean-Nikolić [JN16]	12	6	3	278	0.126	0.113
New constructs with high level of parallelisation and speed						
LeMac [BBL+24]	12	8	4	514	0.068	0.068
SMAC-1 × 4 (this paper)	12	8	4	590	0.059	—
SMAC-1 × 8 (this paper)	24	16	8	954	0.037	—
SMAC-1 × 16 (this paper)	48	32	16	1005	0.035	—

Table 8: Performance evaluation of complete implementations with various data lengths. Measurements of Le&PetitMac *(a)* **include** *(b)* **exclude** the computation time for subkeys.

Performance in Gbps	*Claimed security*	Length of the message, in bytes						
		2^{18}	2^{16}	2^{14}	2^{12}	2^{10}	2^{8}	2^{6}
GHASH (OSSL 3.0.0)	*96*	213	212	202	153	81	78	22
PetitMac, full$^{(a)}$	*128*	90	88	82	64	35	12	3.4
PetitMac, excl$^{(b)}$	*128*	90	90	89	85	73	47	13.3
LeMac, full$^{(a)}$	*128*	445	396	293	124	38	10	2.6
LeMac, excl$^{(b)}$	*128*	455	431	410	251	104	31	8.2
SMAC-1	*118*	156	155	152	142	113	65	19.1
SMAC-3/4	*152*	109	109	108	104	90	58	18.4
SMAC-1/2	*252*	73	73	72	70	63	44	18.4
SMAC-1×4	*118*	579	564	493	315	134	41	10.6
SMAC-1×8	*118*	925	877	723	415	150	41	10.6

6 Conclusions

In this paper, we presented a new efficient stand-alone MAC scheme based on the processing in the FSM part of the stream cipher family SNOW. The proposal offers a combination of very high speed in software and hardware, a truncatable tag and resistance to nonce misuse. Three concrete versions of SMAC are proposed with different security levels. SMAC can be combined with an encryption scheme in an AEAD mode, with high performance and robust security. Every design choice has been argued for through analysis and simulations. The aggregated variant SMAC-1×n achieves the speed up to 925 Gbps which, to the best of our knowledge, is faster than other polynomial and AES based MACs.

A direction for future work could be to examine the possibility of meaningful security proofs for the construct. For example, one might investigate to what extent the `InitFinal` algorithm with $d = 9$ is indistinguishable from a PRF. If so, this might be extended to proofs for the full construct.

Acknowledgements

We thank the cloud teams of E2C and LUNARC for help in computing resources for our simulations that made these results possible. We also thank John Preuß Mattsson, Erik Thormarker, and anonymous reviewers for providing constructive comments. This work was supported by the Swedish Foundation for Strategic Research (Grants No. RIT17-0005 and SM22-0050) and the ELLIIT program.

References

[3GP] 3GPP. 3GPP confidentiality and integrity algorithms. https://www.3gpp.org/specifications-technologies/specifications-by-series/confidentiality-algorithms.

[BBL+24] Augustin Bariant, Jules Baudrin, Gaëtan Leurent, Clara Pernot, Léo Perrin, and Thomas Peyrin. Fast AES-Based Universal Hash Functions and MACs: Featuring LeMac and PetitMac. *IACR Transactions on Symmetric Cryptology*, 2024(2):35–67, Jun. 2024.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 1–15. Springer, Berlin, Heidelberg, August 1996.

[BDH+17] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Farfalle: parallel permutation-based cryptography. *IACR Trans. Symm. Cryptol.*, 2017(4):1–38, 2017.

[BdSF+22] Alex Biryukov, Luan Cardoso dos Santos, Daniel Feher, Vesselin Velichkov, and Giuseppe Vitto. Automated truncation of differential trails and trail clustering in ARX. In Riham AlTawy and Andreas Hülsing, editors, *SAC 2021*, volume 13203 of *LNCS*, pages 286–307. Springer, Cham, September / October 2022.

[Ber05] Daniel J. Bernstein. The poly1305-AES message-authentication code. In Henri Gilbert and Helena Handschuh, editors, *FSE 2005*, volume 3557 of *LNCS*, pages 32–49. Springer, Berlin, Heidelberg, February 2005.

[BHK+99] John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, and Phillip Rogaway. UMAC: Fast and secure message authentication. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 216–233. Springer, Berlin, Heidelberg, August 1999.

[BJKS94] Jürgen Bierbrauer, Thomas Johansson, Gregory Kabatianskii, and Ben Smeets. On families of hash functions via geometric codes and concatenation. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 331–342. Springer, Berlin, Heidelberg, August 1994.

[BÖS11] Joppe W. Bos, Onur Özen, and Martijn Stam. Efficient hashing using the AES instruction set. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 507–522. Springer, Berlin, Heidelberg, September / October 2011.

[BRS02] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 320–335. Springer, Berlin, Heidelberg, August 2002.

[DR05] Joan Daemen and Vincent Rijmen. A new MAC construction ALRED and a specific instance ALPHA-MAC. In Henri Gilbert and Helena Handschuh, editors, *FSE 2005*, volume 3557 of *LNCS*, pages 1–17. Springer, Berlin, Heidelberg, February 2005.

[Dwo07] Morris Dworkin. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. NIST Special Publication 800-38D, November 2007. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf.

[EJMY19] Patrik Ekdahl, Thomas Johansson, Alexander Maximov, and Jing Yang. A new SNOW stream cipher called SNOW-V. *IACR Trans. Symm. Cryptol.*, 2019(3):1–42, 2019.

[GKM+09] Praveen Gauravaram, Lars R Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S Thomsen. Grøstl-a sha-3 candidate. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2009.

[HE22] Hosein Hadipour and Maria Eichlseder. Autoguess: A tool for finding guess-and-determine attacks and key bridges. In Giuseppe Ateniese and Daniele Venturi, editors, *ACNS 22International Conference on Applied Cryptography and Network Security*, volume 13269 of *LNCS*, pages 230–250. Springer, Cham, June 2022.

[Hel80] Martin Hellman. A cryptanalytic time-memory trade-off. *IEEE transactions on Information Theory*, 26(4):401–406, 1980.

[HII+22] Akinori Hosoyamada, Akiko Inoue, Ryoma Ito, Tetsu Iwata, Kazuhiko Mimematsu, Ferdinand Sibleyras, and Yosuke Todo. Cryptanalysis of Rocca and feasibility of its security claim. *IACR Trans. Symm. Cryptol.*, 2022(3):123–151, 2022.

[HK15] Matthias Hamann and Matthias Krause. Stream cipher operation modes with improved security against generic collision attacks. Cryptology ePrint Archive, Report 2015/757, 2015.

[HK18] Matthias Hamann and Matthias Krause. On stream ciphers with provable beyond-the-birthday-bound security against time-memory-data tradeoff attacks. *Cryptography and Communications*, 10(5):959–1012, 2018.

[IK03] Tetsu Iwata and Kaoru Kurosawa. OMAC: One-key CBC MAC. In Thomas Johansson, editor, *FSE 2003*, volume 2887 of *LNCS*, pages 129–153. Springer, Berlin, Heidelberg, February 2003.

[JN16] Jérémy Jean and Ivica Nikolic. Efficient design strategies based on the AES round function. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 334–353. Springer, Berlin, Heidelberg, March 2016.

[LPS21] Gaëtan Leurent, Clara Pernot, and André Schrottenloher. Clustering effect in simon and simeck. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part I*, volume 13090 of *LNCS*, pages 272–302. Springer, Cham, December 2021.

[Mat95] Mitsuru Matsui. On correlation between the order of S-boxes and the strength of DES. In Alfredo De Santis, editor, *EUROCRYPT'94*, volume 950 of *LNCS*, pages 366–375. Springer, Berlin, Heidelberg, May 1995.

[MWGP12] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In Chuan-Kun Wu, Moti Yung, and Dongdai Lin, editors, *Information Security and Cryptology*, pages 57–76, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[NFI24] Yuto Nakano, Kazuhide Fukushima, and Takanori Isobe. Encryption algorithm Rocca-S, 2024. https://datatracker.ietf.org/doc/draft-nakano-rocca-s/.

[Nik14] Ivica Nikolić. Tiaoxin-346. submission to CAESAR competition, 2014. https://competitions.cr.yp.to/round3/tiaoxinv21.pdf.

[NIS] NIST. The Galois/Counter Mode of Operation (GCM). https://csrc.nist.rip/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf.

[oST01] National Institute of Standards and Technology. Advanced encryption standard. *NIST FIPS PUB 197*, 2001.

[PD] Laurent Perron and Frédéric Didier. CP-SAT (v9.9). https://developers.google.com/optimization/cp/.

[SGL+17] Siwei Sun, David Gerault, Pascal Lafourcade, Qianqian Yang, Yosuke Todo, Kexin Qiao, and Lei Hu. Analysis of AES, SKINNY, and others with constraint programming. *IACR Trans. Symm. Cryptol.*, 2017(1):281–306, 2017.

[SII24] Kosei Sakamoto, Ryoma Ito, and Takanori Isobe. Parallel SAT framework to find clustering of differential characteristics and its applications. In Claude Carlet, Kalikinkar Mandal, and Vincent Rijmen, editors, *SAC 2023*, volume 14201 of *LNCS*, pages 409–428. Springer, Cham, August 2024.

[SLN+21] Kosei Sakamoto, Fukang Liu, Yuto Nakano, Shinsaku Kiyomoto, and Takanori Isobe. Rocca: An efficient AES-based encryption scheme for beyond 5g. *IACR Trans. Symm. Cryptol.*, 2021(2):1–30, 2021.

[Sta10] Paul Stankovski. Greedy distinguishers and nonrandomness detectors. In Guang Gong and Kishan Chand Gupta, editors, *INDOCRYPT 2010*, volume 6498 of *LNCS*, pages 210–226. Springer, Berlin, Heidelberg, December 2010.

[TIHM17] Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. Cube attacks on non-blackbox polynomials based on division property. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 250–279. Springer, Cham, August 2017.

[Tod15] Yosuke Todo. Structural evaluation by generalized integral property. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 287–314. Springer, Berlin, Heidelberg, April 2015.

[WHT+18] Qingju Wang, Yonglin Hao, Yosuke Todo, Chaoyun Li, Takanori Isobe, and Willi Meier. Improved division property based cube attacks exploiting algebraic properties of superpoly. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 275–305. Springer, Cham, August 2018.

[WP14] Hongjun Wu and Bart Preneel. AEGIS: A fast authenticated encryption algorithm. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 185–201. Springer, Berlin, Heidelberg, August 2014.

[XZBL16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 648–678. Springer, Berlin, Heidelberg, December 2016.

A Permutation candidates

Table 9: Strong permutation candidates found for SMAC-1.

SMAC-1. Minimum number of active S-boxes of a differential trail $(\Delta M^t, ..., \Delta M^{t+\Delta t-1})$ where the first and the last Δs are nonzero.							If used in FSM, min. number of active S-boxes
$\Delta t = 3$	$\Delta t = 4$	$\Delta t = 5$	$\Delta t = 6$	$\Delta t = 7$	$\Delta t = 8$	$\Delta t = 9$	
22	20	20	20	21	[22,23]	[23,25]	18
σ_1 ={0,7,14,11,4,13,10,1,8,15,6,3,12,5,2,9}					σ_2 ={0,9,6,13,4,11,2,15,8,1,14,5,12,3,10,7}		
20	20	19	19	19	20	$\geq$22	18
σ_3 ={4,9,2,13,0,11,6,15,12,1,14,5,8,3,10,7}					σ_4 ={4,9,6,13,0,11,2,15,12,1,10,5,8,3,14,7}		
σ_5 ={8,1,6,13,12,11,2,15,0,9,14,5,4,3,10,7}					σ_6 ={8,7,14,3,12,13,10,1,0,15,6,11,4,5,2,9}		
σ_7 ={8,7,14,11,12,5,10,1,0,15,6,3,4,13,2,9}					σ_8 ={8,9,6,13,12,11,2,7,0,1,14,5,4,3,10,15}		
σ_9 ={12,7,2,11,8,13,10,1,4,15,6,3,0,5,14,9}					σ_{10} ={12,7,14,11,8,13,6,1,4,15,10,3,0,5,2,9}		
24	20	19	20	22	24	$\geq$25	17
σ_{11} ={7,10,5,8,11,14,9,12,15,2,13,0,3,6,1,4}					σ_{12} ={13,8,15,10,1,12,3,14,5,0,7,2,9,4,11,6}		
20	20	19	19	19	20	22	16
σ_{13} ={4,1,14,11,0,13,10,7,12,15,6,3,8,5,2,9}					σ_{14} ={4,7,14,11,0,13,10,1,12,9,6,3,8,5,2,15}		
σ_{15} ={12,9,6,3,8,11,2,15,4,1,14,5,0,13,10,7}					σ_{16} ={12,9,6,13,8,5,2,15,4,1,14,11,0,3,10,7}		
24	19	19	20	22	24	$\geq$25	15
σ_{17} ={6,5,15,12,13,9,8,14,3,2,4,7,10,0,1,11}					σ_{18} ={9,5,4,10,15,14,0,3,6,12,13,7,2,1,11,8}		
σ_{19} ={11,10,12,15,2,8,9,3,14,13,7,4,5,1,0,6}					σ_{20} ={14,4,5,15,10,9,3,0,1,13,12,2,7,6,8,11}		

Table 10: Clusters of differential trails of SMAC-1. In this table, b is the number of byte-trails, c is the number of clusters, p is the probability of the cluster, and s is the minimum number of active S-boxes.

Case		A few selected permutations from Table 9 for SMAC-1				
		σ_1	σ_3	σ_{11}	σ_{13}	σ_{17}
$\Delta t = 3$	$b : c\ (s)$	4:4 (22)	2:2 (20)	16:16 (24)	2:2 (20)	16:16 (24)
	p	2^{-134}	2^{-121}	2^{-152}	2^{-121}	2^{-153}
$\Delta t = 4$	$b : c\ (s)$	2496:192 (20)	534:192 (20)	60:44 (20)	382:80 (20)	2:2 (19)
	p	$2^{-121.21}$	$2^{-120.44}$	$2^{-124.41}$	$2^{-120.83}$	2^{-122}
$\Delta t = 5$	$b : c\ (s)$	4032:288 (20)	256:68 (19)	900:140 (19)	248:64 (19)	866:130 (19)
	p	$2^{-121.25}$	$2^{-115.67}$	$2^{-119.41}$	$2^{-115.88}$	$2^{-118.40}$
$\Delta t = 6$	$b : c\ (s)$	18321:96* (20)	(19)	2352:232 (20)	(19)	(20)
	p	$2^{-118.95}$	—	$2^{-123.56}$	—	—
$\Delta t = 7$	$b : c\ (s)$	30264:46** (21)	(19)	(22)	(19)	(22)
	p	$2^{-129.27}$	—	—	—	—
Max.	p	$\mathbf{2^{-118.95}}$	$2^{-115.67}$	$\mathbf{2^{-119.41}}$	$2^{-115.88}$	$\mathbf{2^{-118.40}}$

* *Only these clusters are found in practical time.*

** *Only trails that have up to 24 active S-boxes were enumerated.*

Comments on simulation details for the clustering effect, relates to Table 10 and Table 12. We skipped testing some scenarios for certain permutations since the maximum probability was already larger than some other permutation candidate had at the time of simulations, and thus unnecessary simulations on the already worse case would only take resources with no influence on the final result. Also, some heavy cases such as $(\Delta t = 6, k = 2)$ and $(\Delta t = 7, k = 1, 5)$ for SMAC-3/4 were also skipped, since all remained "good" permutations have at least 27 and 31 active S-boxes in their best byte-trails for these scenarios, respectively, which is already much larger than the minimum 24 and therefore

would not cross the maximum probability that is already detected by testing other cases of the same permutation. Finally, for a few cases where $\Delta t \geq 6$, the models in OR-Tools became too big and the number of byte-trails was huge, so only those clusters with the highest chance to break through the security level were checked thoroughly.

Comments on the search of the candidates for SMAC-3/4, relates to Table 11. We did not include the filter for $\Delta t = 4$ in the SMAC-3/4 search since the performance for this filter at high thresholds wasn't fast enough. Also, with the threshold of 26 we initially hoped to get a minimum of 25 active S-boxes overall (also for $\Delta t > 3$), but additional analysis and simulations showed that there are no permutations that have a minimum of 25 active S-boxes. Then we settled for targeting 24 active S-boxes, while already having 1.6 B candidates from the first round, and received the short list of promising ones among these candidates having at least 24 active S-boxes in $\Delta t > 3$. Thus, a separate round of first-phase heavy simulations for $\Delta t = 3$ and threshold 24 could be done, but we assess it would not be advantageous.

Table 11: Strong permutation candidates found for SMAC-3/4.

SMAC-3/4. Minimum number of active S-boxes of a differential trail $(\Delta M^t, ..., \Delta M^{t+\Delta t-1})$ where the first and the last Δs are nonzero, encountering cases with dummy middle clock(s), i.e. where $\Delta M^{t+k} = 0$.							If used in FSM, minimum number of active S-boxes
$\Delta t = 3$	$\Delta t = 4$ $k = 1/2$	$\Delta t = 5$ $k = 1/2/3$	$\Delta t = 6$ $k = 2/3$	$\Delta t = 7$ $k = 1, 5$ $/3$	$\Delta t = 8$ $k = 1, 5$ $/2, 6$	$\Delta t = 9$ $k = 1, 5$ $/2, 6/3, 7$	
26	24/30	24/24/24	24/24	29/25	27/33	≥29/29/35	20
σ_{21} ={7,9,0,12,15,8,2,13,6,1,14,11,5,3,10,4}				σ_{22} ={9,7,14,8,11,13,4,0,3,12,6,1,10,5,2,15}			
σ_{23} ={11,4,14,9,2,13,10,7,1,15,6,0,3,5,12,8}				σ_{24} ={14,9,6,3,13,11,2,12,15,1,8,4,7,0,10,5}			
26	26/32	24/26/26	26/≥24	32/25	30/≥37	30/35/≥35	18
σ_{25} ={4,2,10,11,0,13,14,6,12,7,9,5,8,3,15,1}				σ_{26} ={4,15,1,13,0,11,7,9,12,10,2,3,8,5,6,14}			
σ_{27} ={8,3,6,5,12,7,10,9,0,11,14,13,4,15,2,1}				σ_{28} ={8,15,14,1,12,3,2,5,0,7,6,9,4,11,10,13}			
σ_{29} ={12,7,3,5,8,6,14,15,4,1,2,10,0,11,13,9}				σ_{30} ={12,9,10,2,8,3,5,1,4,15,11,13,0,14,6,7}			
26	35/30	24/≥25/24	≥24/24	30/25	28/35	31/34/≥35	17
σ_{31} ={4,9,11,6,8,13,15,10,12,1,3,14,0,5,7,2}				σ_{32} ={4,13,14,7,12,11,15,10,0,6,1,5,8,9,2,3}			
σ_{33} ={8,7,11,6,12,2,13,1,4,5,14,15,0,9,10,3}				σ_{34} ={8,14,9,13,0,1,10,11,12,5,6,15,4,3,7,2}			
σ_{35} ={12,13,6,7,8,1,2,11,0,15,3,14,4,10,5,9}				σ_{36} ={12,14,9,11,0,2,13,15,4,6,1,3,8,10,5,7}			
26	26/28	24/25/25	27/24	31/24	27/37	30/35/≥34	17
σ_{37} ={4,7,12,15,8,13,10,9,0,1,3,14,2,6,5,11}				σ_{38} ={4,8,6,10,12,11,14,15,1,5,13,7,0,9,2,3}			
σ_{39} ={4,9,6,5,12,13,15,10,14,2,1,7,0,3,8,11}				σ_{40} ={4,13,6,7,8,12,10,14,0,15,2,3,5,9,1,11}			
σ_{41} ={6,10,9,15,8,11,0,3,12,1,14,13,4,5,7,2}				σ_{42} ={7,14,15,10,12,13,3,0,4,6,1,5,8,11,2,9}			
σ_{43} ={8,7,10,11,13,1,9,3,12,5,14,15,0,4,2,6}				σ_{44} ={8,9,10,13,0,14,2,12,4,5,6,15,7,1,11,3}			
σ_{45} ={8,9,11,6,10,14,13,3,12,15,4,7,0,5,2,1}				σ_{46} ={8,9,15,12,0,2,13,1,4,7,14,5,3,10,11,6}			
σ_{47} ={8,14,9,11,0,3,2,15,4,13,12,5,10,1,7,6}				σ_{48} ={9,13,5,15,8,1,10,11,12,0,14,2,4,3,6,7}			
σ_{49} ={11,5,15,7,12,13,14,1,4,2,6,0,8,9,10,3}				σ_{50} ={12,5,4,13,2,9,15,14,0,6,1,3,8,11,10,7}			
σ_{51} ={12,10,14,8,0,1,2,11,3,13,7,15,4,5,6,9}				σ_{52} ={12,13,14,7,15,9,3,11,0,1,2,5,8,6,10,4}			
σ_{53} ={12,14,9,13,0,3,10,1,15,6,7,2,4,5,11,8}				σ_{54} ={12,15,6,13,11,2,3,14,0,1,7,4,8,10,5,9}			
σ_{55} ={12,15,14,11,0,9,8,1,6,13,3,2,4,10,5,7}				σ_{56} ={14,5,11,10,12,2,13,15,4,7,6,3,8,1,0,9}			
26	26/26	24/24/25	24/24	30/26	29/32	31/31/35	16
σ_{57} ={4,15,5,11,14,1,8,2,12,7,13,6,9,3,0,10}				σ_{58} ={4,15,5,14,1,11,8,2,12,7,13,3,6,9,0,10}			
σ_{59} ={10,13,4,14,8,3,9,2,5,15,12,6,0,11,1,7}				σ_{60} ={13,7,4,14,8,3,9,15,2,5,12,6,0,11,1,10}			

Table 12: Cluster characteristics of differential trails of SMAC-3/4.

Case		A few selected permutations from Table 11 for SMAC-3/4								
		σ_{21}	σ_{25}	σ_{27}	σ_{31}	σ_{32}	σ_{36}	σ_{37}	σ_{42}	σ_{57}
$\Delta t = 3$	b	39	44	44	8	8	8	12	12	38
	c	24	28	28	8	8	8	12	12	32
	p	$2^{-159.67}$	2^{-162}	$2^{-162.41}$	2^{-163}	2^{-163}	2^{-163}	2^{-162}	2^{-163}	$2^{-160.67}$
S-boxes	#	26	26	26	26	26	26	26	26	26
$\Delta t = 4$	b	1/2	4/4	4/4	236/4	236/4	236/4	4/4	4/4	4/5
	c	1/1	4/4	4/4	212/4	212/4	212/4	4/1	4/1	4/2
$k=1$	p	2^{-151}	2^{-163}	2^{-164}	$2^{-218.41}$	$2^{-216.41}$	$2^{-218.41}$	2^{-162}	2^{-163}	2^{-161}
$k=2$	p	$2^{-195.41}$	2^{-208}	2^{-208}	2^{-194}	2^{-192}	2^{-193}	$2^{-178.19}$	$2^{-179.67}$	$2^{-167.83}$
S-boxes	#	24/30	26/32	26/32	35/30	35/30	35/30	26/28	26/28	26/26
$\Delta t = 5$	b	1158	2583	2616	2888	3172	3272	171	155	796
		/3/7	/52/342	/98/304	/102/12	/8/17	/102/8	/5/4	/5/4	/14/203
	c	40/2/4	90/19/4	100/32/4	48/36/4	52/1/8	52/36/4	20/5/1	20/5/1	27/4/25
$k=1$	p	$2^{-146.23}$	$2^{-145.57}$	$2^{-146.08}$	$2^{-139.75}$	$2^{-139.27}$	$2^{-137.83}$	$2^{-151.37}$	$2^{-152.29}$	$2^{-146.02}$
$k=2$	p	$2^{-149.67}$	$2^{-164.91}$	$2^{-165.09}$	$2^{-165.67}$	$2^{-160.41}$	$2^{-164.67}$	2^{-164}	2^{-165}	$2^{-152.54}$
$k=3$	p	$2^{-157.41}$	$2^{-159.06}$	$2^{-152.39}$	$2^{-156.54}$	2^{-156}	$2^{-156.83}$	$2^{-153.83}$	$2^{-154.83}$	$2^{-149.21}$
S-boxes	#	24/24/24	24/26/26	24/26/26	24/26/24	24/25/24	24/26/24	24/25/25	24/25/25	24/24/25
$\Delta t = 6$	b	4/191						–/316	–/264	1333/112
	c	1/12	—	—	—	—	—	–/45	–/35	14/6
$k=2$	p	$2^{-159.83}$						–	–	$2^{-147.86}$
$k=3$	p	$2^{-145.79}$						$2^{-151.21}$	$2^{-152.86}$	$2^{-149.04}$
S-boxes	#	24/24	26/25	26/24	24/24	24/24	25/24	27/24	27/24	24/24
$\Delta t = 7$	b							–/5*	–/6*	
	c	—	—	—	—	—	—	–/2	–/1	—
$k=1,5$	p							–	–	
$k=3$	p							$2^{-156.60}$	$2^{-156.83}$	
S-boxes	#	29/25	32/25	32/25	30/25	30/25	30/25	31/24	31/24	30/26
Max.	p	$2^{-145.79}$	$2^{-145.57}$	$2^{-146.08}$	$2^{-139.75}$	$2^{-139.27}$	$2^{-137.83}$	$\mathbf{2^{-151.21}}$	$\mathbf{2^{-152.29}}$	$2^{-146.02}$

* *Only significant trails that have at most 57 active S-boxes were enumerated.*

In this table, b is the number of byte-trails, c is the number of clusters, p is the probability of the cluster, and value(s) in k mean that $\Delta M^{t+k} = 0$ are dummy blocks.

Table 13: Strong permutation candidates found for SMAC-1/2, and the analysis results of the clustering effect. In this table, #S-boxes is minimum number of active S-boxes of a differential trail $(\Delta M^t, ..., \Delta M^{t+\Delta t-1})$ where the first and the last Δs are nonzero, encountering cases with dummy middle clock(s), i.e. where $\Delta M^{t+k} = 0$. For clustering effect, b is the number of byte-trails; c is the number of clusters; and p is the upper bound for all clusters' probabilities. For all permutations, if used in FSM, the minimum number of active S-boxes in a linear approximation is 18.

SMAC-1/2. Permutation, σ	$\Delta t = 5, k = 1, 3$		$\Delta t = 7, k = 1, 3, 5$	
	$b : c$	p	$b : c$	p
#S-boxes: **41** in $\Delta t = 5, k = 1, 3$; **41** in $\Delta t = 7, k = 1, 3, 5$; **49** in $\Delta t = 9, k = 1, 3, 5, 7$				
Clustering effect analysis for $\sigma_{\mathbf{61}}$	2838:266	$2^{-256.22}$	15656:174*	$2^{-252.30}$
σ_{61} ={0,11,7,14,6,4,1,15,9,3,8,5,13,2,10,12}	σ_{62} ={1,6,14,0,4,15,11,2,10,8,5,3,13,7,12,9}			
σ_{63} ={1,11,0,13,5,10,2,4,8,3,15,6,14,12,9,7}	σ_{64} ={2,0,13,11,5,15,4,1,9,14,6,8,12,7,3,10}			
* *only significant trails with at most 51 active S-boxes were enumerated for* $\Delta t = 7$.				
	2854:267	—	—	—
σ_{65} ={0,6,13,9,7,4,10,2,11,15,8,1,14,5,3,12}	σ_{66} ={2,9,7,0,4,10,1,13,11,8,14,6,15,3,12,5}			
σ_{67} ={3,0,6,14,7,11,4,13,10,1,15,8,12,2,9,5}	σ_{68} ={3,7,0,9,6,13,11,4,8,14,5,1,15,12,2,10}			
#S-boxes: **41** in $\Delta t = 5, k = 1, 3$; **43** in $\Delta t = 7, k = 1, 3, 5$; **49** in $\Delta t = 9, k = 1, 3, 5, 7$				
Clustering effect analysis for $\sigma_{\mathbf{69}}$	4510:782	$2^{-256.00}$	—	—
σ_{69} ={0,5,15,10,14,8,1,13,7,3,12,6,11,2,9,4}	σ_{70} ={10,4,13,9,3,15,8,2,7,14,5,0,12,1,11,6}			
σ_{71} ={15,6,13,8,4,9,3,14,2,12,5,1,11,7,0,10}	σ_{72} ={15,11,4,14,3,10,1,12,8,13,7,2,6,0,9,5}			
	4600:794	—	—	—
σ_{73} ={0,10,5,15,9,12,11,2,13,14,4,1,6,7,3,8}	σ_{74} ={5,6,12,9,14,15,11,0,8,2,13,7,1,4,3,10}			
σ_{75} ={5,8,7,14,9,10,0,13,2,3,15,4,12,6,1,11}	σ_{76} ={10,11,7,12,4,14,9,3,13,0,15,6,1,2,8,5}			

B On a binary differential trail for SMAC-1, $\Delta t = 3$

We actually wrote our own program in C++ with the highest possible optimisation for the case of SMAC-1, $\Delta t = 3$ (as well as for a few other cases). The reason was that if we aim to test 2^{44} permutations then this filtering stage must be very fast. To comment, the results that our custom program produced were double-verified by the MILP model and CP tools, which we further used for $\Delta t > 3$ cases.

Let us first derive the state expression after 3 clocks:

$$\text{1st clock:} \begin{cases} A1^{t+1} = \sigma(A2^t \oplus A3^t \oplus M^t) \\ A2^{t+1} = L\pi S(A1^t) \oplus M^t \\ A3^{t+1} = L\pi S(A2^t) \oplus M^t \end{cases}$$

$$\text{2nd clock:} \begin{cases} A1^{t+2} = \sigma(L\pi S(A1^t) \oplus L\pi S(A2^t) \oplus M^{t+1}) = x \oplus \sigma M^{t+1} \\ A2^{t+2} = L\pi S(\sigma(A2^t \oplus A3^t \oplus M^t)) \oplus M^{t+1} = L\pi S(y \oplus \sigma M^t) \oplus M^{t+1} \\ A3^{t+2} = L\pi S(L\pi S(A1^t) \oplus M^t) \oplus M^{t+1} = L\pi S(z \oplus M^t) \oplus M^{t+1} \\ \text{where: } x = \sigma(L\pi S(A1^t) \oplus L\pi S(A2^t)),\ y = \sigma(A2^t \oplus A3^t),\ z = L\pi S(A1^t) \end{cases}$$

$$\text{3rd clock:} \begin{cases} A1^{t+3} = \sigma(L\pi S(y \oplus \sigma M^t) \oplus L\pi S(z \oplus M^t) \oplus M^{t+2}) \\ A2^{t+3} = L\pi S(x \oplus \sigma M^{t+1}) \oplus M^{t+2} \\ A3^{t+3} = L\pi S(L\pi S(y \oplus \sigma M^t) \oplus M^{t+1}) \oplus M^{t+2} \end{cases}$$

We want to find a differential $\Delta(M^t, M^{t+1}, M^{t+2})$ such that $\Delta(A1, A2, A3)^{t+3} = 0$, and after the substitutions $\Delta M^{t+2} \to L\pi\Delta M^{t+2}$, $\Delta M^{t+1} \to \sigma^{-1}\Delta M^{t+1}$, $\Delta M^t \to \sigma^{-1}\Delta M^t$,

the system is simplified to

$$\begin{cases} \Delta M^{t+2} = S(y \oplus \Delta M^t) \oplus S(z \oplus \sigma^{-1}\Delta M^t) \\ \Delta M^{t+2} = S(x \oplus \Delta M^{t+1}) \\ \Delta M^{t+2} = S(L\pi S(y \oplus \Delta M^t) \oplus \sigma^{-1}\Delta M^{t+1}) \end{cases}$$

The attacker starts introducing the first differential in time t, and can introduce up to 3 consecutive differentials, hoping that the state will recover in time $t+3$. Note that, since the compression function Π is reversible, the internal state may only collide in that certain time $t+3$, and not later.

From the second equation it is clear that $\mu(\Delta M^{t+2}) = \mu(\Delta M^{t+1})$, and thus that equation can be virtually removed from consideration as follows:

$$\begin{cases} \Delta M^{t+2} = S(y \oplus \Delta M^t) \oplus S(z \oplus \sigma^{-1}\Delta M^t) \\ \Delta M^{t+2} = S(L\pi S(y \oplus \Delta M^t) \oplus \sigma^{-1}S^{-1}(x \oplus \Delta M^{t+2})) \end{cases}$$

In a naïve approach it is obviously possible just to loop over all 16-bit masks $\mu(\Delta M^t)$ and $\mu(\Delta M^{t+2})$ in time 2^{32} and check for feasibility of the above system to have a solution.

In practice, we note that L operates on the groups of 4 bytes independently (now in these byte-trails encoded as just 4 bits), so that when we test a new permutation candidate, we can bring the above differential relation into four slices each of 4 bits width – this makes it possible to create a precomputed table that contains whether a binary trail for these 4-bit slice may exist or not feasible. This way, once in the loop we have chosen $\mu(\Delta M^t)$ and $\mu(\Delta M^{t+2})$, the check for feasibility may be done almost instantly.

Table 14: An example trail for SMAC-1 in $\Delta t = 3$.

Clocks	Variables	Intermediate differences
0	$\Delta A1$	00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
	$\Delta A2$	00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
	$\Delta A3$	00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
1	$\Delta S(A1)$	00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
	$\Delta S(A2)$	00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
	ΔM^t	80,00,00,00,00,00,59,00,00,0e,00,00,00,84,00,00
	$\Delta A1$	80,00,00,00,00,84,00,00,00,00,59,00,00,00,00,0e
	$\Delta A2$	80,00,00,00,00,00,59,00,00,0e,00,00,00,84,00,00
	$\Delta A3$	80,00,00,00,00,00,59,00,00,0e,00,00,00,84,00,00
2	$\Delta S(A1)$	39,00,00,00,00,a8,00,00,00,00,01,00,00,00,00,71
	$\Delta S(A2)$	11,00,00,00,00,00,64,00,00,40,00,00,00,01,00,00
	ΔM^{t+1}	ed,00,00,00,00,44,40,00,00,32,24,00,00,ac,00,10
	$\Delta A1$	ed,00,00,00,00,ac,24,00,00,10,40,00,00,44,00,32
	$\Delta A2$	0c,00,00,00,00,44,40,00,00,32,24,00,00,ac,00,10
	$\Delta A3$	cf,11,11,33,c0,c4,00,40,03,30,25,01,64,00,c8,74
3	$\Delta S(A1)$	28,00,00,00,00,a8,64,00,00,40,01,00,00,01,00,71
	$\Delta S(A2)$	28,00,00,00,00,a8,64,00,00,40,01,00,00,01,00,71
	ΔM^{t+2}	c3,11,11,33,c0,80,40,40,03,02,01,01,64,ac,c8,64
	$\Delta A1$	00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
	$\Delta A2$	00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
	$\Delta A3$	00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00

If a solution to the system is feasible, then the total number of active S-boxes is [9]:

$$\#\text{ S-boxes} = 2 \cdot [HW(\mu(\Delta M^t)) + HW(\mu(\Delta M^{t+2}))]$$

We can also include that metric into the truncation of the loop in the way we only loop through $\mu(\Delta M^t)$ and $\mu(\Delta M^{t+2})$ such that the number of active S-boxes is strictly less than the already found best value.

For SMAC-1 with σ_1 and $\Delta t = 3$ we found the following byte differential whose trail has 22 active S-boxes, which is the minimum in this scenario. Here, $\mu(\Delta M^{t+2})$ is the value before the aforementioned substitutions on ΔMs.

$$\begin{aligned}
\mu(\Delta M^t) &= (1,0,0,0,0,0,1,0,0,1,0,0,0,1,0,0)\\
\mu(\Delta M^{t+1}) &= (1,0,0,0,0,1,1,0,0,1,1,0,0,1,0,1)\\
\mu(\Delta M^{t+2}) &= (1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
\end{aligned}$$

One of many possible *bit-level* differential vectors that correspond to the above *byte* differential is given in Table 14.

C Altered instantiations of SMAC

Depending on the use case, the model can be altered. For example, when SMAC is used in AEAD mode paired with an encryption algorithm, the initialisation phase can be skipped by assigning the internal state in time $t = 0$ with three pseudo-random secret values produced by the accompanying cipher.

There could be various other use cases for the SMAC framework, and the exact instances we have provided here is partly to give concrete security bounds and advice on sufficient number of dummy clocks in the initialisation and finalisation phases. If an application needs a different security/performance trade-off, the number of clocks during the initialisation/finalisation phase may be changed.

Minimum d. Depending on the use case, required performance, security demands, the tag size etc., we advise the minimum number of rounds for the initialisation phase must be $d_{\texttt{Init}} \geq 6$ (if not combined with an external cipher), and for the finalisation phase it must be $d_{\texttt{Final}} \geq 4$ rounds (the first time when the tag is influenced by all registers). These absolute minimums are supported by the results of our analyses in Section 3.

C.1 Example of using SMAC-1 with AES in AEAD mode

Assume we are using AES-256-CTR with an IV value consisting of 12 bytes of nonce and domain separation and 4 bytes of counter value, starting at zero with IV_0. The subscript of IV indicates the counter value. The first three keystream symbols are:

$$\begin{aligned}
Z^0 &= \text{AES-256}_K(IV_0)\\
Z^1 &= \text{AES-256}_K(IV_1)\\
Z^2 &= \text{AES-256}_K(IV_2)
\end{aligned}$$

where $\text{AES-256}_K(P)$ denotes the application of AES-256 on the 16 byte plaintext array P using the key K. These values can be directly loaded into the SMAC registers $(A1, A2, A3)$ and compression of the messages (AAD and ciphertext) can start immediately. This scheme is depicted in Figure 6.

[9] Note that for $\Delta t = 3$ the number of active S-boxes is always an even number. I.e., if we filter for 26 active S-boxes and do not find permutations giving at least 25 in other $\Delta t > 3$, then the next step down would be $\Delta t = 3$ with 24 active S-boxes.

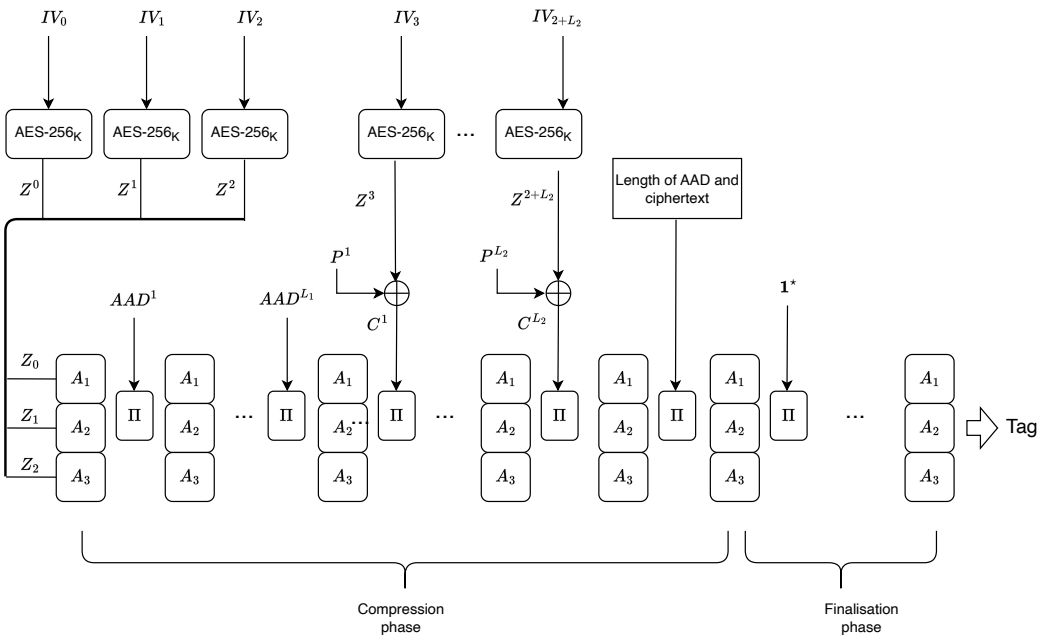

Figure 6: Example of SMAC usage as AEAD integrity protection together with AES-256-CTR.

D A fast heuristic algorithm to find a small guess base in guess-and-determine attack scenarios

The main idea on describing relations between variables comes from [HE22], but since Autoguess works very slow already for $d \geq 3$, we decided to develop a simplified yet powerful enough tool to solve GnD systems where all variables have the same weight (weight 1 to all byte variables in our case) and only the basic type of relation supported.

A relation on n variables $[x_0, x_1, \ldots, x_{n-1}]$ is added to the system when the knowledge of any $n-1$ variables results in the knowledge of the remaining unknown in the list. To note, a new relation and new variables are added to the system only at the points of branching. In our case, we have two such points.

Relations for `XOR`: Let us have a branching point such as $c = a \oplus b$, then the relation here is simply $[a, b, c]$, meaning that the knowledge of any 2 values (bytes) would result in the knowledge of the third value. Note that here a new variable c is introduced into the system.

Relations for `MixColumn`: Consider a 4-by-4 `MixColumn` operation from AES. The input is four existing variables (x_0, x_1, x_2, x_3), and the output are new variables (y_0, y_1, y_2, y_3) to be added to the system. The `MixColumn` linear transformation is such that knowing any 4 values from 8 input and output variables would result in the knowledge of all other 4 values. This can be described with 56 5-tuple relations (8 choose 5) such as $[x_0, x_1, x_2, x_3, y_0], \ldots, [y_0, y_1, y_2, y_3, x_3]$.

Application of S-boxes and permutations of the array of variables do not create any new relation nor introduce any new variable. This way, the complete system comprises a set of v variables (some of which can be set as known) and a set of r relations between these variables, and that system can be described by a *binary* matrix R of size $r \times v$, e.g.:

$$R_{r,v} = \begin{array}{cccc} x_0 & x_1 & ... & x_{v-1} \\ \hline 1 & 0 & ... & 1 \\ 1 & 1 & ... & 0 \\ \vdots & & & \\ 0 & 1 & ... & 1 \end{array}$$

where v columns represent variables and r rows are relations. Introduce the following v-bit vectors:

- K_v – the vector of variables that are known from the start, such as observed output bytes or those bytes where we insert IV, which are known, etc.
- G_v – the vector of the guess base, initialised as $G_v = \mathbf{0}$.

The target of the solver is to find G with the minimum Hamming weight such that given only variables from the set $(K \vee G)$ one can derive all other variables in time $O(1)$ by using the relation matrix R.

Let us now introduce a **knowledge propagation function:** $F_{KP}(R_{r\times v}, K_v) \to K'_v$, which is, given the relation matrix R and a vector of all known variables at the moment K, derives a new vector of known variables K' after applying the matrix with relations R. This function works as follows:

Algorithm 5 Knowledge propagation function.

1: **function** $F_{KP} : (R, K) \to K'$
2: Set $K' = K$
3: **for** all $i = 0, 1, \ldots, r-1$ **do**
4: **if** Hamming weight of $(\texttt{NOT}(K') \wedge R.\texttt{row}(i))$ is 1 **then**
5: $K' = K' \vee R.\texttt{row}(i)$
6: If at least one bit was added to K' during the above **for-loop**, repeat that loop again until no more new bits can be added to K'.

The algorithm of finding the guess base consists of two phases – the *Approximation* and *Reduction* phases, as briefly described below. The algorithm is a variation of a greedy approach, but comparing to Autogess it works extremely fast and still gives quite good results. Since it is still a heuristic algorithm, one should expect that the resulting guess base may not be optimal, but hopefully close to the minimum.

Approximation phase. We start by computing $Y = F_{KP}(R, K \vee G)$, and then also remove rows $R.\texttt{row}(i)$ from R where Hamming weight of $Y \wedge R.\texttt{row}(i)$ is zero – i.e., these relations become not helpful in the GnD flow.

Then, in each step of this phase we try all unknown variables one by one (those where Y is '0'), and collect **metrics** for each of these unknowns – we will talk about various metrics further. The unknown variable with the best metric is added to the guess base G, and Y is updated as $Y = F_{KP}(Y \vee x)$, while also removing rows from R that in this step became covered by Y.

The phase ends when the Hamming weight of Y becomes equal to v, i.e., all variables became known.

In an improved variant each step we test all possible pairs of unknown variables and the one with best metrics is added to the guess base G, and Y is updated with two points added. Testing a triple-point is more costly but still feasible time. However, we used the 3-points method only on few analysis cases.

Metrics. Let us pick one unknown variable x that is not in Y. We have identified two main metrics:

(a) the Hamming weight of $F_{KP}(R, Y \vee x)$ – the larger Hamming weight the more variables become known if that particular x is added to the guess base.

(b) the Hamming weight of the column of (the truncated) R corresponding to x – the more 1s are removed from R, the more new variables may be derived through the knowledge propagation.

There can be any order of (a) and (b) metrics for the decision which candidate for the base guess is better to adopt, and we have tried both orders in our simulations and finally took the shorted guess base from both methods. In case of a tie-break decision, we apply additional metrics:

(c) choose the best candidate between two equal options at random

(d) prioritise the candidate involving the unknown variable closer to other known variables – i.e., in case of SMAC analysis we prefer to avoid guessing variables somewhere in the middle of d rounds of `InitFinal`.

Reduction phase. After an approximate guess base is received, we then start the last phase of reduction of the base. We simply try to remove two guessed variables from the guess base, and see whether adding one other unknown would still give a valid guess base. In an improved variant, one may also remove 3 variables, try to add 1 or 2 other unknowns and check if the guess base is still valid. But this appeared to bee too timely and thus we did not use 3-points reduction.

E Estimate of $\#\Delta out$ for a single stream

For every fixed Δin and Δout, the number of dominating characteristics over $(n_1 + n_2)$ rounds is upper bounded by c. In order to compute the total number of dominating characteristics, we need to estimate the number of valid Δout, applicable to any fixed input differential.

Let us revert the last linear operations on Δout until the last application of S-boxes. I.e., if $\Delta out = \Delta(A1^t, A2^t, A3^t)$ for $t = (n_1 + n_2)$ then the differential without the ending linear operations is $\Delta out' = \Delta(A1', A2', A3')$ and derived as

$$\begin{aligned} A1' &= S(A1^{t-2}) \oplus S(A2^{t-2}) \\ A2' &= S(A1^{t-1}) \\ A3' &= S(A2^{t-1}) \end{aligned}$$

which are expressed via $(A1^{t-1}, A1^{t-2}, A2^{t-1}, A2^{t-2})$, such that

$$\begin{aligned} A1^t &= \sigma(L \cdot A1' \oplus \mathbf{1}^\star) \\ A2^t &= L \cdot A2' \oplus \mathbf{1}^\star \\ A3^t &= L \cdot A3' \oplus \mathbf{1}^\star \end{aligned}$$

is a linear mapping between $\Delta out'$ and Δout, i.e., $\#\Delta out' = \#\Delta out$.

Assume $\Delta out'$ has exactly k active bytes, then the number of possible $\Delta out'$ is less than $\binom{48}{k} \cdot 2^{8k}$ (each nonzero byte may have one of $(2^8 - 1)$ values). Let $\mathtt{wt}(X)$ be the number of nonzero bytes in a binary vector X, i.e. $\mathtt{wt}(X) = HW(\mu(X))$. The number of active bytes in $\Delta out'$ cannot be more than the number of active S-boxes in those four registers, i.e.,

$$\begin{aligned} k &= \mathtt{wt}(\Delta A1') + \mathtt{wt}(\Delta A2') + \mathtt{wt}(\Delta A3') \\ &\leq \mathtt{wt}(\Delta S(A1^{t-1})) + \mathtt{wt}(\Delta S(A1^{t-2}) + \mathtt{wt}(\Delta S(A2^{t-1})) + \mathtt{wt}(\Delta S(A2^{t-2})) = k' \end{aligned}$$

For the dominating characteristics, the number k' can not be larger than the maximum number of active S-boxes contained in the last two rounds $\kappa = \omega[n_1, n_2] - \omega[n_1, n_2 - 2]$, i.e. $k' \leq \kappa$, since $\Delta out'$ involves only registers and S-boxes from the last two rounds. Then we get $k \leq k' \leq \kappa$, and the range of k is $k \in [1..\kappa]$. Thus, the total number of valid Δout can be upper bounded by

$$\#\Delta out < \sum_{k=1}^{\omega[n_1,n_2]-\omega[n_1,n_2-2]} \binom{48}{k} \cdot 2^{8k}$$

F Reference implementation (C/C++, SIMD)

```
#define SMAC_VER 1  /* SMAC instance: {1,34,12} for SMAC-{1,3/4,1/2} resp.*/
#define SIGMA (SMAC_VER == 1 ?\
      _mm_setr_epi8(0,7,14,11,4,13,10,1,8,15,6,3,12,5,2,9)   /* SMAC-1   */\
    : (SMAC_VER == 34?\
      _mm_setr_epi8(7,14,15,10,12,13,3,0,4,6,1,5,8,11,2,9)   /* SMAC-3/4 */\
    : _mm_setr_epi8(0,11,7,14,6,4,1,15,9,3,8,5,13,2,10,12))) /* SMAC-1/2 */

#define load(ptr)      _mm_loadu_si128((__m128i*)(ptr))
#define store(ptr, x) _mm_storeu_si128((__m128i*)(ptr), x)
#define aes(a, k)      _mm_aesenc_si128(a, k)
#define sigma(x)       _mm_shuffle_epi8(x, SIGMA)
#define xor2(x, y)     _mm_xor_si128(x, y)
#define xor3(x, y, z)  xor2(xor2(x,y),z)

void SMAC_Compress(__m128i& A1, __m128i& A2, __m128i& A3, uint8_t* msg)
{       __m128i M = msg ? load(msg) : _mm_cvtsi32_si128(1);
        __m128i T = sigma(xor3(A2, A3, M));
        A3 = aes(A2, M);
        A2 = aes(A1, M);
        A1 = T;
}

void SMAC_InitFinal(__m128i& A1, __m128i& A2, __m128i& A3)
{       __m128i T1 = A1, T2 = A2, T3 = A3;
        for (int i = 0; i < 9; i++)
                SMAC_Compress(A1, A2, A3, NULL);
        A1 = xor2(A1, T1);
        A2 = xor2(A2, T2);
        A3 = xor2(A3, T3);
}

// (!) In this implementation, aad/ct must reserve 16/32 extra bytes, resp.
void SMAC(uint8_t key[32], uint8_t iv[16], uint8_t* aad, int aad_sz,
          uint8_t * ct, int ct_sz, uint8_t * tag, int tag_sz)
{
        // initialise with the key and iv
        __m128i A1 = load(key + 16), A2 = load(key), A3 = load(iv);
        SMAC_InitFinal(A1, A2, A3);

        // zeroise ending unaligned bytes, and add LEN-block
        memset(aad + aad_sz, 0, 16);
        memset(ct + ct_sz, 0, 16);
        int aad_blocks = (aad_sz + 15) >> 4;
        int ct_blocks = (ct_sz + 15) >> 4;
        *(uint64_t*)(ct + (ct_blocks * 16) + 0) = aad_sz * 8;
        *(uint64_t*)(ct + (ct_blocks * 16) + 8) = ct_sz * 8;

        // compress full blocks, including the ending LEN-block to ct
        for (int i = 0; i <= (aad_blocks + ct_blocks); i++)
        {       uint8_t* msg = i < aad_blocks ? (aad + i * 16)
                                : (ct + (i - aad_blocks) * 16);
                SMAC_Compress(A1, A2, A3, msg), num_clocks++;
                if (SMAC_VER == 12 || (SMAC_VER == 34 && (i % 3) == 2))
                        SMAC_Compress(A1, A2, A3, NULL);
        }

        // finalise and derive the MAC value
        SMAC_InitFinal(A1, A2, A3);
        memcpy(tag, (uint8_t*)&A2, (tag_sz <= 16 ? tag_sz : 16));
        if (tag_sz > 16)
```

```
                memcpy(tag + 16, (uint8_t*)&A3, tag_sz - 16);
}
```

Listing 2: Reference implementation of SMAC-{1, 3/4, 1/2} in C/C++.

G Test vectors

The MAC tag is taken as $(A2||A3)_\tau$ after the finalisation phase.

```
=== TEST 1 ===
      KEY = { 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
              00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 }
       IV = { 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 }
      AAD = { }
   CIPHER = { }
For SMAC-1:
    After initialisation:
        A1 = { fa 4e 8b ba 5b a3 79 be 90 a7 ee d8 00 12 03 5b }
        A2 = { 8c 99 e7 01 95 ba 79 b6 e1 3f 0f 56 6a d4 5c 60 }
        A3 = { 59 ec 45 58 1d a5 08 9e e4 ad 8e 4d e2 da b1 08 }
    After compression (1 clock):
        A1 = { d5 28 ed 1b 88 0e 81 75 05 68 71 59 88 1f a2 92 }
        A2 = { 55 78 3c 27 19 f8 94 9f 13 00 3a 13 60 9d 98 fe }
        A3 = { 69 dd 17 95 fd 62 4f b9 e9 81 51 53 2a b5 53 27 }
    After finalisation:
        A1 = { aa 8c 58 31 e0 ce 87 91 08 b7 c2 63 1e 2e 9b f9 }
        A2 = { d8 2c 49 ea 46 81 ca 1f ba 97 93 49 5f 9a 60 85 }
        A3 = { 39 ce be 86 12 c8 0f 70 60 cf 18 41 2e 98 92 ee }
For SMAC-3/4:
    After initialisation:
        A1 = { 10 34 48 ab 43 0d ac c5 e1 b8 38 03 ed 27 fe 80 }
        A2 = { 7c 90 c3 d8 c9 55 eb 3a 83 98 a1 5f 92 30 fb 56 }
        A3 = { ae 25 80 ee 48 1a bd 5e b7 73 2f 62 a9 a6 ed 37 }
    After compression (1 clock):
        A1 = { 64 16 61 8e 3b 96 36 d2 81 56 b5 4f 34 3d 43 eb }
        A2 = { 27 bb 5f 14 59 76 bd 3d 50 2b 61 da 68 b6 f9 80 }
        A3 = { bc 14 40 87 05 21 26 f7 61 16 2f 1e 18 60 ac dd }
    After finalisation:
        A1 = { df a6 f5 9c 06 06 36 cf b5 85 9d 4c a5 ca bc f7 }
        A2 = { 66 49 62 35 b1 7d 4c 42 2c ce 5f 42 9d 45 6c 91 }
        A3 = { 3f 41 13 bc 6d 27 65 ac bb 5e 83 72 ca 99 41 f1 }
For SMAC-1/2:
    After initialisation:
        A1 = { 3a a2 52 41 82 fa 64 14 23 2e b7 fb f7 14 b6 76 }
        A2 = { 57 a4 aa dd 99 2e 1e c7 44 c7 82 b8 51 8d c5 c5 }
        A3 = { 3e 72 05 09 7f 54 e2 9d 30 e9 92 6c 0d f8 ea e2 }
    After compression (2 clocks):
        A1 = { ac 34 87 0d 07 49 a8 54 bb f7 18 69 6f 11 9d ae }
        A2 = { b9 19 96 98 7c 70 48 63 9b d6 60 17 d5 d6 57 ad }
        A3 = { 28 c3 fb b1 88 a8 1b 9a df 0e 28 cf 8d 4b da bb }
    After finalisation:
        A1 = { 88 49 95 3a 08 db 76 e3 4a dc c6 af c7 78 cc d0 }
        A2 = { 67 06 22 e0 2a d6 85 85 b9 90 4c 1c 8f 33 45 51 }
        A3 = { 7d 2b d8 95 62 6d 99 dd 40 c9 34 d9 85 13 3f 64 }

=== TEST 2 ===
      KEY = { 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
              00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 }
       IV = { 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 }
      AAD = { 03 }
   CIPHER = { }
For SMAC-1:
    After initialisation:
        A1 = { 42 e3 c1 df bd 96 9f a1 04 02 9a 30 c4 93 fa 26 }
        A2 = { 3a fc c4 b7 10 45 50 5e b1 d1 09 49 f9 d6 de 13 }
        A3 = { 25 c5 ff c2 ab 2b 5b 15 62 43 fc f0 e5 6a be 33 }
    After compression (2 clocks):
        A1 = { 61 dc 6e d4 a7 60 66 11 04 c3 c0 a7 5e 45 be a8 }
        A2 = { e0 64 7e 6f b9 f4 78 dc b4 3a 74 c1 96 4d 44 cb }
        A3 = { 48 34 ed 24 58 af a3 e2 9d 2e 4c ac 5b 10 07 52 }
    After finalisation:
        A1 = { 13 0e 94 2c 5b 1f 89 23 5e c6 9a c0 77 f6 9c 91 }
        A2 = { a1 35 23 df 28 37 ed d8 0f 6b 56 aa 61 17 80 b3 }
        A3 = { 8a 7b 4b e4 8f 4b 4b de b7 d5 af 8c 82 6d 81 6d }
For SMAC-3/4:
```

```
    After initialisation:
        A1 = { a3 1a 8c d8 b9 c6 d7 24 d4 9b 5b 75 ff 67 41 64 }
        A2 = { 5f db ff 2f c9 aa f4 3e 32 ef f5 a9 ff 07 42 33 }
        A3 = { 1e eb df 0b eb e4 70 6b b8 3f f6 da cf 73 cf 24 }
    After compression (2 clocks):
        A1 = { 0a b0 36 58 d2 b0 88 ee 90 99 0f 98 e0 9c e3 f9 }
        A2 = { 32 bf f6 69 25 03 a3 12 6b b1 93 89 02 b1 3e b7 }
        A3 = { 39 57 c5 65 51 50 6a a6 c6 b8 8c 8f ea 46 eb 84 }
    After finalisation:
        A1 = { f2 33 7e b0 54 87 37 5b 6e f6 f3 64 67 07 93 80 }
        A2 = { 39 bf fe 0e 2c 33 11 f7 51 69 8e 64 d0 4e 52 70 }
        A3 = { c0 99 5e 83 54 a5 a8 22 57 94 06 c0 49 f2 0a 6f }
For SMAC-1/2:
    After initialisation:
        A1 = { e2 fc c5 64 a8 dd ed c1 53 57 50 cb a4 e4 15 7d }
        A2 = { 60 e1 3c 40 92 1c 80 0c 91 dc 4c 1f b8 59 e6 d3 }
        A3 = { 3e 18 80 f9 9a 46 60 fb b1 c3 7f ca 59 e6 7c 39 }
    After compression (4 clocks):
        A1 = { 25 c0 5f db 33 85 76 df c7 22 96 2f e2 fb 4d 36 }
        A2 = { 10 b2 5a 50 03 32 3d 61 19 b8 39 16 e2 48 ff f2 }
        A3 = { ad a1 eb 31 ff f9 e6 39 c9 7d dd 72 3b fe 90 66 }
    After finalisation:
        A1 = { 79 ee 8b fa 25 19 39 7d 64 f7 6a ec e6 9e 3f bb }
        A2 = { e0 a3 33 94 3d 50 cd 2c 31 6d f0 a5 b6 4b 76 21 }
        A3 = { 70 87 5c 28 5d 9b 39 be 56 4f 6b 9a 7a 0a d1 e8 }

=== TEST 3 ===
       KEY = { b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 ba bb bc bd be bf
               c0 c1 c2 c3 c4 c5 c6 c7 c8 c9 ca cb cc cd ce cf }
        IV = { d0 d1 d2 d3 d4 d5 d6 d7 d8 d9 da db dc dd de df }
       AAD = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 ea eb ec dd de ef }
    CIPHER = { f0 f1 f2 f3 f4 f5 f6 f7 f8 f9 fa fb fc fd fe ff }
For SMAC-1:
    After initialisation:
        A1 = { db 1d 65 de 28 12 23 17 15 8d ab 00 04 5f 22 5c }
        A2 = { e1 11 bc 2b c5 47 d0 19 15 83 6f 95 1f 47 5f 84 }
        A3 = { 00 91 7c c9 66 f0 f3 84 07 b2 6d 48 cf 7c 06 f8 }
    After compression (3 clocks):
        A1 = { 0f 0f 18 4b 4a 3a 25 0a ef b3 82 01 ce 2c 59 9c }
        A2 = { 0f 1a 78 a3 a9 a9 00 63 e8 21 f0 ed 82 52 80 12 }
        A3 = { 56 b8 b1 7f 40 bf a9 16 e9 5a 19 9f dd b9 98 60 }
    After finalisation:
        A1 = { a9 d2 6c f8 c3 75 b6 6f b5 28 d3 e2 80 75 b8 cc }
        A2 = { 61 3f ad 89 9e 94 51 48 1a eb d1 7a 5c 64 dd 18 }
        A3 = { 9a c4 ac 2e 18 74 a4 e1 cf 9b 42 92 15 38 a9 a1 }
For SMAC-3/4:
    After initialisation:
        A1 = { 30 b2 8e a9 d7 6b 44 d1 74 21 21 c5 68 43 45 62 }
        A2 = { 84 79 59 30 73 11 5b a2 bd 12 a3 85 66 66 43 20 }
        A3 = { c6 56 7a de ff 9f 3e fa a0 fb a4 6f f2 73 b8 d3 }
    After compression (4 clocks):
        A1 = { 13 5b 81 4d 81 50 f1 cf 5a cf 7b cf e5 1e b0 7c }
        A2 = { 72 13 2e cf 8b 8a f1 54 0c f2 8b 27 c4 66 b8 0d }
        A3 = { 75 3b de a8 94 36 d3 da 52 49 e6 17 8c 92 78 7c }
    After finalisation:
        A1 = { 98 d5 fe f2 0c e2 c7 4d 74 2a ed b1 25 81 3e da }
        A2 = { db 13 1c b3 ff bc a2 ed ae a4 78 93 58 18 67 5a }
        A3 = { 6b b8 f5 a9 83 7b c5 9f 4d 45 fd a7 60 31 cf 53 }
For SMAC-1/2:
    After initialisation:
        A1 = { b5 eb ed ac 6b bd 4d ab 56 23 a6 ce 3b 0e dc 0e }
        A2 = { b4 a2 75 44 a1 ac 33 d0 a7 96 f2 ff 3f ce c3 cd }
        A3 = { 5b 81 96 5a ad 89 aa c2 28 54 a3 8c 43 f7 15 c9 }
    After compression (6 clocks):
        A1 = { f1 2a 8a 10 99 80 d7 bf ff 6d e3 e1 cf 4b 6d 22 }
        A2 = { c0 16 59 c7 eb 5f b5 44 4c f5 27 82 b6 4c 42 e8 }
        A3 = { c4 de c3 46 fe b3 e3 19 13 61 48 bf 3a 89 10 7f }
    After finalisation:
        A1 = { 4a b7 f0 63 c8 60 7f ca 08 4b 27 bb 1f 4a dc 70 }
        A2 = { 80 ea 3b d5 07 42 40 bd 8e 66 ae 69 68 99 99 e7 }
        A3 = { 53 e2 de f4 17 c1 6f 53 c4 ce c2 73 37 74 e9 3b }

=== TEST 4 ===
       KEY = { 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
               10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f }
        IV = { ff fe fd fc fb fa f9 f8 f7 f6 f5 f4 f3 f2 f1 f0 }
       AAD = { 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10
               11 12 13 }
```

```
    CIPHER = { 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 }
For SMAC-1:
    After initialisation:
        A1 = { 5c f1 48 92 aa 70 1c 6c 0f 7b 8d 57 96 0c 39 4b }
        A2 = { ee 31 08 25 85 30 fc 59 8e a6 c3 ec 57 2f dc 59 }
        A3 = { 45 cc 6d 77 d2 56 28 a9 be 38 5a 78 4b a1 ba 14 }
    After compression (4 clocks):
        A1 = { 50 62 2a 64 fc 70 1b 1d 8e 6d 9f 12 dc f5 b7 7c }
        A2 = { d8 9d 7b 14 68 94 59 74 51 74 60 c7 56 d2 16 3f }
        A3 = { 45 73 4e 18 8e d3 4d ae 78 31 d3 59 6b fa 47 c7 }
    After finalisation:
        A1 = { 82 98 b1 ab 90 54 76 e4 24 76 b3 78 d6 14 e8 08 }
        A2 = { c3 44 52 16 99 48 2d 93 28 3c 03 ec 7c 3d b8 b5 }
        A3 = { c7 77 64 62 16 89 98 ee 28 03 06 f9 25 33 09 7c }
For SMAC-3/4:
    After initialisation:
        A1 = { b7 1a 78 eb a6 e1 a2 02 6f 0b 87 2d f3 82 29 93 }
        A2 = { 06 46 fe a4 94 d8 20 18 e3 3d 52 b3 bd b7 19 5e }
        A3 = { 34 55 a2 94 e7 11 e2 10 cc b8 89 fb c9 98 29 6d }
    After compression (5 clocks):
        A1 = { fa 9e 30 f3 39 72 e3 0b c3 57 f3 49 1f 76 cc c3 }
        A2 = { cb db bb df 38 4f 34 f1 ef 48 fd 7f d3 1f 7d a7 }
        A3 = { e9 cd ed 82 6b eb 7e e2 20 db 2f df 34 bf 8e 55 }
    After finalisation:
        A1 = { 84 9c ca a1 1b 55 64 ba 15 72 b2 b9 0d 73 ba d3 }
        A2 = { 69 6e d0 a9 9e 04 84 3a 59 6d a5 b6 25 7d db de }
        A3 = { 65 6d 19 04 1d bb 04 58 35 c3 42 3b c4 92 61 4f }
For SMAC-1/2:
    After initialisation:
        A1 = { 06 55 0f dc 34 89 87 c6 52 d3 cf 05 65 fb 6a 6a }
        A2 = { fc fb 2c 8d 45 a3 0d 79 26 88 fd fa e2 ca 7d 06 }
        A3 = { 45 3d 86 5e 1e 98 85 c5 0c 39 09 71 bb 99 c2 c0 }
    After compression (8 clocks):
        A1 = { 11 15 96 5f 0d 8f e9 d3 56 e6 4b 5d d6 4c 2a c7 }
        A2 = { 96 e8 fa 24 65 d4 aa 40 73 8e 62 77 12 98 ff 2b }
        A3 = { 69 12 b5 35 47 3e f8 00 eb 95 c3 7e dc 13 25 30 }
    After finalisation:
        A1 = { 81 db 76 2f 94 f8 c1 34 bb 37 48 2a fc 0f 0f 4f }
        A2 = { f7 c3 6b bf 83 44 90 6e 17 ca cb 97 0e 37 50 26 }
        A3 = { dc 06 99 2b 75 0e 08 66 f6 54 79 ed d0 2e 3c a4 }
```

Listing 3: Test vectors.