Fast AES-Based Universal Hash Functions and MACs

Augustin Bariant, Jules Baudrin, Gaëtan Leurent, Clara Pernot, Léo Perrin, Thomas Peyrin

IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2024, No. 2, pp. 35–67. DOI:10.46586/tosc.v2024.i2.35-67

Fast AES-Based Universal Hash Functions and MACs

Featuring LeMac **and** PetitMac

Augustin Bariant[1,2], Jules Baudrin[1], Gaëtan Leurent[1], Clara Pernot[1], Léo Perrin[1] and Thomas Peyrin[3]

[1] Inria, Paris, France
name.surname@inria.fr
[2] ANSSI, Paris, France
[3] Nanyang Technological University, Singapore, Singapore
thomas.peyrin@ntu.edu.sg

Abstract. Ultra-fast AES round-based software cryptographic authentication/encryption primitives have recently seen important developments, fuelled by the authenticated encryption competition CAESAR and the prospect of future high-profile applications such as post-5G telecommunication technology security standards. In particular, Universal Hash Functions (UHF) are crucial primitives used as core components in many popular modes of operation for various use-cases, such as Message Authentication Codes (MACs), authenticated encryption, wide block ciphers, etc.
In this paper, we extend and improve upon existing design approaches and present a general framework for the construction of UHFs, relying only on the AES round function and 128-bit word-wide XORs. This framework, drawing inspiration from tweakable block ciphers design, allows both strong security arguments and extremely high throughput. The security with regards to differential cryptanalysis is guaranteed thanks to an optimized MILP modelling strategy, while performances are pushed to their limits with a deep study of the details of AES-NI software implementations. In particular, our framework not only takes into account the number of AES-round calls per message block, but also the very important role of XOR operations and the overall scheduling of the computations.
We instantiate our findings with two concrete UHF candidates, both requiring only 2 AES rounds per 128-bit message block, and each used to construct two MACs. First, LeMac, a large-state primitive that is the fastest MAC as of today on modern Intel processors, reaching performances of 0.068 c/B on Intel Ice Lake (an improvement of 60% in throughput compared to the state-of-the-art). The second MAC construction, PetitMac, provides an interesting memory/throughput tradeoff, allowing good performances on many platforms.

Keywords: Universal hash function · MAC · AES · authentication

1 Introduction

Since its standardization, the AES block cipher [DR02] has deeply influenced the design of symmetric-key cryptographic primitives. This trend even accelerated after the introduction in modern CPUs of AES-NI [Gue08], a set of dedicated hardware accelerated instructions implementing the AES encryption and decryption. To benefit from that potential performance boost, designers continued studying operating modes allowing a direct and efficient reuse of the full AES [MV04, RBB03, KR21]. Yet, since AES-NI granularity lies at the

Received: 2024-03-01 Accepted: 2024-05-01 Published: 2024-06-18

round level, many new cryptographic designs actually use the AES round-function as building block, either for hash functions [BBG+08, IAC+08, BD08, GK08], for authenticated encryption schemes [WP14, Nik14, JNPS21, SLN+21, NFI24], for permutations [IIL+23, GM16, KLMR16, BLLS22], or collision resistant building blocks [JN16, Nik17a], among other applications. Today, hardware acceleration of the AES round-function is widespread in most computing platforms, from high-end Intel/AMD CPUs to microcontrollers for mobile devices, and AES-NI have become even more helpful over the processors generations, with reduced latency and increased throughput.

These technological advances allowed many symmetric-key primitives to eventually reach throughput performances under 1 c/B, but new use-cases arise. In particular, sixth-generation mobile communication systems (6G) plan to deliver transmissions with an impressive throughput range of 100 Gbps to 1 Tbps. This puts a lot of pressure on the encryption/authentication performances and AES-NI-based solutions seem very natural. This is the direction taken by the Authenticated Encryption (AE) algorithm Rocca [SLN+21, SLN+22] and its updated version Rocca-S [NFI24], currently the fastest AE on AES-NI platforms and under submission at IETF. Recently, the round function framework of Rocca has been further analysed in a work that presents optimal round function candidates (in terms of speed) within the framework [TSI23].

More generally, there has been significant efforts to design symmetric primitives relying on AES rounds (and the corresponding processor intrinsics), such as AEGIS [WP14], Tiaoxin [Nik14] or Aerion [BLLS22]. We note that most of these primitives have sub-optimal throughputs on some recent processors. For instance, the optimal candidate in the Rocca round function framework [TSI23] reaches a throughput of 0.104 cycles per byte on Tiger Lake, while the maximum theoretical throughput is 0.0625 cycles per byte for any candidate with the same number of AES rounds per 128-bit message, as explained in Section 2.2.

1.1 Universal Hash Functions and Message Authentication Codes

In this paper, we study the construction of (almost) universal hash functions (UHF) based on AES rounds. UHFs take as input a secret key and a plaintext, and map them to a fixed-length digest. Formally, we consider them as a family of functions indexed by a key (choosing a key corresponds to choosing a member in the family), with two different security notions: almost-universal hash functions (ε-AU), and almost-XOR-universal hash functions (ε-AXU), defined as follows:

Definition 1 (ε-AU)**.** A family of functions $H : A \to B$ is ε-almost-universal if:

$$\forall m \neq m' \in A,\ |\{h \in H : h(m) = h(m')\}| \leq \varepsilon |H|$$

Definition 2 (ε-AXU)**.** A family of functions $H : A \to B$ is ε-almost-XOR-universal if:

$$\forall m \neq m' \in A,\ \forall d \in B,\ |\{h \in H : h(m) \oplus h(m') = d\}| \leq \varepsilon |H|$$

The ε-AU notion only requires collision resistance on average over a random key. The ε-AXU notion is a stronger variant to cover an arbitrary output difference, rather than just collisions. In particular, if H is an ε-AXU family, it is also an ε-AU family.

UHF security notions are relatively weak, so that they can be fulfilled by purely combinatorial constructions. However, they are quite versatile; in particular, a UHF can be turned into a Message authentication Code (MAC) with a few extra components.

UHF-based MACs. A MAC also processes a message and a secret key to generate a tag (a nonce-based MAC would also take as input a non-repeating nonce value), but in order to ensure authenticity/integrity of the message, a stronger security notion is expected. It

should be hard for an attacker to construct a forgery on a MAC, i.e. generating a valid combination of message/tag without knowledge of the secret key.

More formally, for a key K, a nonce N and a message M, a nonce-based MAC F consists of a signing algorithm $\mathrm{AUTH}_K(M, N)$ that generates a tag T, and a verification algorithm $\mathrm{VER}_K(M, N, T)$ that returns "valid" if $\mathrm{AUTH}_K(M, N) = T$ and "invalid" otherwise. A (q, v, t)-adversary against the nonce-based MAC-security of F is an adversary $\mathcal{A}$ with access to oracles AUTH_K and VER_K, making at most q MAC queries to the AUTH_K oracle, at most v verification queries to VER_K oracle, and running in time at most t. We say that $\mathcal{A}$ forges if any of its queries to VER_K returns "valid". The advantage of $\mathcal{A}$ against the nonce-based MAC security of F is defined as

$$\mathbf{Adv}_F^{\mathrm{MAC}}(\mathcal{A}) = \Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathrm{AUTH}_K, \mathrm{VER}_K} \text{ forges}]$$

where $K \xleftarrow{\$} \mathcal{K}$ denotes that K is chosen uniformly at random from the set $\mathcal{K}$ of possible keys and where $\mathcal{A}$ is not allowed to ask a verification query (N, M, T) to VER_K if a previous query (N, M) to AUTH_K returned T. Note that $\mathcal{A}$ is also not allowed to repeat nonces for AUTH_K, but can repeat them for VER_K.

MACs are classically built from block ciphers, but also from UHF. Notably, GMAC [Dwo07] and Poly1305 [Ber05] are two popular MACs based on UHF which use polynomial evaluation in a finite field as a UHF. They use the Wegman-Carter-Shoup construction [CW77, Sho96] to construct a nonce-based MAC from UHF. However, it only provides $2^{n/2}$ security for a n-bit tag with unique nonces, and fails completely when nonces are repeated. The EWCDM construction [CS16] guarantees a significantly higher security as it was proven [MN17] to provide essentially 2^n security with unique nonces and even $2^{n/2}$ when nonces are repeated.

Arithmetic UHFs. There has been a significant effort to design fast UHF based on arithmetic operations: Polynomial hashing (GHASH used in GCM [MV04], Poly1305 [Ber05]), NH in UMAC [BHK+99], etc. These constructions can be quite fast, and have a proven security level. For instance, GHASH only requires a single multiplication and a single addition in $\mathbb{F}_{2^{128}}$ for every 128-bit block of plaintext. This is particularly interesting in environments where instructions enabling fast arithmetic in the finite field of size 2^{128} are provided, which is the case of most modern processors intended for a usage in servers and desktop computers. On the other hand, Poly1305 and UMAC rely on integer multiplication.

AES-based UHFs. Dedicated design strategies for block ciphers and hash functions are well known, but dedicated Universal Hash Functions (UHFs) have received less attention. In particular, processors that enable fast computation of arithmetic for UHFs often support fast computation of a full AES round as well. Therefore, in this paper we focus on designing fast UHFs based on the AES round.

In this context, we note the interesting PC-MAC construction of Minematsu and Tsunoo [MT06]. Based on the analysis of the the Maximum Expected Differential Probability (MEDP) of 4-round AES by Keliher and Sui [KS07], they consider 4-round AES as an ε-AXU family with $\varepsilon \approx 1.18 \cdot 2^{-110}$ (under the hypothesis that the round keys are independent). Using this as a building block, they construct a MAC with 4 AES rounds per 128-bit block of plaintext, with provable security. Another interesting work is the EliMAC primitive proposed by Dobrauning *et al.* [DMN23], which uses 11 AES rounds per 128-bit message block (7 rounds can be precomputed in an offline phase, leaving 4 in the online phase).

We thus aim for fewer than 4 AES rounds per block of message, but the achieved security will be heuristic, instead of relying on a formal security proof.

1.2 Our contributions

In this paper, we present a family of UHFs than can reach better performances than state-of-the-art UHFs, by exploiting the extremely high throughput of AES-NI instructions, with flexible parameters that can be adapted to future computing platforms. We select candidates from our family with no differential trail of probability higher than 2^{-128}, suggesting that these UHFs are ε-AU UHFs with $\varepsilon \approx 2^{-128}$. Unlike most ε-AU UHFs, the ε-AU property of our candidates is not proved but rather heuristic, as we only ensure that no high probability differential trails exist. Our construction uses a novel design strategy compared to previous UHFs or collision resistant round functions, with a (potentially large) internal state separated into two parts: one part updated with non-linear and linear components (the AES round function and 128-bit XORs in our case), influenced by another part updated with linear components only (this second part is not influenced by the first one to reduce dependencies that would complicate both the instructions scheduling and the automated security analysis). Several fresh message blocks are inserted within the second part at each round so as to ensure a low rate. The general idea is that while a large state indeed complicates the attacker's task, updating it entirely can be costly and partial updates might lead to better security/performance tradeoffs. Thus, this separation strategy offers more granularity and draws inspiration from recent (tweakable) block cipher designs, where the tweakey schedule is linear, or Panama [DC98] hash function.

Although this family is too big to be exhausted in practice, we propose a process to iterate over candidates of the family and select some fast and secure ones. We implemented a tool that, given any candidate of this family, automatically computes the number of active S-boxes in the best differential trail, using Mixed Integer Linear Programming (MILP). The MILP model exactly discards linear incompatibilities, improving on heuristic approaches to avoid linear incompatibilities [CHP+17]. In addition, our tool can compile on-the-fly candidates of the family and benchmark them, in order to automatically measure their speed. To our knowledge, this is the first time that on-the-fly benchmarking is performed to filter candidates in an AES-based framework.

To showcase the relevance of our approach, we present an ε-AU UHF candidate whose round function reaches a speed of 0.067 cycles per byte on Intel Tiger Lake (i5-1135G7). In addition, we show the very first candidate with less than 2 AES rounds per 128-bit message block and 128-bit collision security, namely with 1.75 AES rounds per 128-bit message block.

From ε-AU UHF candidates of our family, we present two new MAC candidates: LeMac and PetitMac. The former is as of today the fastest MAC on modern desktop/server processors, reaching of speed of 0.068 cycles per byte on Intel Ice Lake (Xeon Gold 5320) for 256 kB messages vs. 0.113 cycles per byte for a MAC based on the round function of [JN16], the fastest state-of-the-art MAC according to our benchmarks. PetitMac is slower, but of a smaller size, thus more suitable for lightweight applications. Even though 6G communications (as targeted by Rocca-S [NFI24] AEAD) would mandate 256-bit keys for post-quantum considerations, both our MAC candidates have 128-bit keys, nonces and tags, as we believe this is largely sufficient for most applications, especially for MACs that do not suffer from "harvest now, decrypt later" attack strategies. We claim that LeMac and PetitMac both provide 128-bit security in the nonce-respecting setting.

Outline. We start with a detailed description of our design goals, and their interactions with the state-of-the-art, in Section 2. In light of this discussion, we decided to focus our efforts on a specific family of UHFs which we present in Section 3. Since this family is very large, we reduce the search space using for instance equivalence classes (see Section 4), and we automate the security analysis using MILP-based methods described in Section 5. The results of our search are presented in Section 6. Finally, we use these results to build concrete primitives in Section 7, while Section 8 concludes the paper.

2 Design Goals and First Observations

While several AES-based constructions exist, we identified places where there remains substantial room for improvement. Below, we describe the goals that our family of UHFs are intended to fulfil; the family itself will be described in Section 3.

2.1 AES-based round functions

As already mentioned in introduction, many designs rest upon the AES round function and the 128-bit XOR to be both secure and efficient, thanks to the AES-NI instructions set in modern processors. Among them, the CAESAR candidates Tiaoxin [Nik14] and AEGIS [WP14] (the latter was selected in the final high-performance portfolio) are competitive AEAD schemes. In terms of throughput, they are outperformed by the building blocks designed by Jean & Nikolić [JN16] and later Nikolić [Nik17a]. Recently, the AEAD proposals Rocca [SLN+21, SLN+22] and Rocca-S [NFI24] target 6G requirements in terms of speed and security. All of those constructions aim at minimizing the so-called *rate* [JN16], that is, the number of AES rounds per 128-bit message block. Rocca (during Additional Data processing) and one of the schemes of Jean & Nikolić achieve a rate of 2 for 128-bit security. We will adopt a similar strategy and minimize the rate of the round function.

Goal 1. *Our ε-AU families should use AES rounds as internal components for high software performance, and preferably at the lowest rate.*

2.2 Instruction scheduling

Modern processors are superscalar processors with out-of-order execution. They can execute several instructions simultaneously, and schedule instructions as soon as the input operands are ready. Moreover the execution units are pipelined: some instructions take several cycle to process, and the execution unit can start processing a new instruction at every clock cycle, with the output being ready some cycles later [Int24].

There are two main metrics to measure the performance of an instruction I:

Latency: the number of clock cycles between the beginning of I to the return of its result. We denote $L(I)$ the latency of I.

Throughput: the number of instructions that can be processed in a given amount of time. We usually consider the reciprocal throughput, measured in cycles. We denote $T(I)$ the throughput of I.

Processors are composed of several execution units, accessed by ports denoted $P_1 \dots P_k$. Each port P_i accepts a certain set of instructions S_i. At cycle t, each execution unit P_i can process an instruction $I \in S_i$, and returns its result $L(I)$ cycles later. At cycle $t+1$, the execution unit P_i might process another instruction $I' \in S_i$ (with potentially $I' = I$), even though the instruction I of cycle t has not returned its result yet. The throughput of an instruction I corresponds to the number of ports which can process I.

For the AES-based ciphers mentioned in Section 2.1, two types of instructions are extensively used: AES rounds instructions (e.g. AESENC), and 128-bit XORs. Note that in recent processors one can leverage 512-bit instructions (like VAESENC that can process four AES rounds in parallel), but in this work we focus on 128-bit instructions that are now widely available. This setting constitutes a very fair comparison with previous schemes and we can expect that most AES-based designs will greatly benefit from more AES rounds in parallel.

Each instruction has its own throughput and latency on modern processors [Fog22], but we cannot exploit the full throughput of both types of instructions at the same time, because they share ports, as illustrated by Table 1. In particular, on modern Intel processors (Ice

Table 1: Scheduling of AESENC and XOR instructions on modern processors [Fog22].

Architecture	Instr	Latency	Throughput	P_0	P_1	P_2	P_3	P_4	P_5	P_6
Intel Haswell	XOR	1	0.33	x	x				x	
	AESENC	7	1						x	
Intel Skylake	XOR	1	0.33	x	x				x	
	AESENC	4	1	x						
Intel Ice Lake	XOR	1	0.33	x	x				x	
	AESENC	3	0.5	x	x					
Intel Tiger Lake	XOR	1	0.33	x	x				x	
	AESENC	3	0.5	x	x					
AMD Zen 1/2/3/4	XOR	1	0.25	x	x	x	x			
	AESENC	4	0.5	x	x					

Lake and higher) and AMD processors (Zen1 and higher), AES-NI instructions operate on two ports P_0 and P_1, while the 128-bit XOR operates on three or four: P_0, P_1, P_5 (or P_0, P_1, P_2, P_3). This implies that rate-2 constructions (*i.e.* with 2 AES-NI instructions per 128-bit message block) require at least 1 cycle per 128-bit message, which corresponds to at least 0.0625 cycles per byte. On the other hand, a rate-2 construction might reach this bound of 0.0625 cycles per byte on these processors, if the pipelining is favorable.

On the number of XOR instructions in AES-based constructions. In the case of Intel Processors (Ice Lake and higher), the throughput of AESENC encryption is 0.5, and the throughput of XOR is 0.33; AESENC requires ports 0 or 1, and XOR requires ports 0, 1, or 5. In order to fully exploit the throughput of the AESENC instruction, we need to feed ports 0 and 1 only with AESENC instructions at each clock cycle, thus they become unavailable for XOR instructions, and XOR instructions can only be assigned to port 5. Consequently, if x AESENC instructions are executed at full throughput, there should be at most $x/2$ XOR instructions. Similar observations apply to recent processors, and unfortunately, minimizing the number of XOR of AES-based constructions is not a systematic approach[1]. For example, Jean & Nikolić [JN16] present rate-2 candidates with 6 AES per rounds and 9 128-bit XORs, and Rocca's rate-2 round function uses 4 128-bit XORS and 4 AES rounds instructions. As a consequence, regardless of implementation tricks, a full throughput on current modern Intel processors will always remain out of reach for these algorithms.

Dependency chains. In addition to the throughput analysis, dependency chains affect the performance of AES-based constructions (see Section 3.2.2 of [Int24]). As an example, let us denote x_i the first wire of an AES-based construction at round i. If x_{i+1} depends on x_i, the latency to compute x_{i+1} from x_i is the sum of the latency of each involved instruction. Then, the latency of the round function is at least the latency of computing x_{i+1} from x_i. In the decryption mode of Rocca [SLN+21], we found a cycle of dependency with 6 cycles of latency, whose latency even increases to 8 cycles in practice on Ice Lake processors. This is explained in Appendix A, and we believe that it leads to a maximal theoretical speed of 0.25 cycles per byte of message on recent Intel processors.

Apart from these two points, there are a lot of processor subtleties, which are difficult to exhaustively consider. As a general guideline, we aim at avoiding any pipelining issue and at being as efficient as possible on modern processors.

Goal 2. *The instruction scheduling in modern processors should be favorable.*

Goal 2 is very reasonable, but is hard to guarantee with pen-and-paper analysis because of the always-evolving, complex, and well-optimized scheduling of modern processors.

[1] To the best of our knowledge, minimizing the number of XOR has only been considered in the permutation design of Gueron and Mouha [GM16], which consider zero XOR instruction outside the AESENC instruction.

In fact, one way to directly evaluate the performance with state-of-the-art instruction scheduling algorithms is to compile and benchmark candidates on-the-fly. This strategy exploits advanced techniques from compilers (e.g. modern gcc) or processors, and remains future-proof, since it can easily be adapted to future processors.

Goal 3. *Our tool should automatize the benchmarking of candidates. The automatic benchmarking should be adaptable to all processors.*

2.3 Security

As already depicted, we design a UHF family with the ε-AU security, and are therefore only interested in collision resistance. UHFs with the security notion of ε-AXU can then be computed from ε-AU UHFs, but that is out of the scope of our ε-AU UHF family.

In order to facilitate the security analysis of our candidates, we consider that the output of one of our ε-AU UHFs is not of a single word, but rather the entire state composed of multiple 128-bit words. In addition, we consider that the inner state is fully unknown, key-dependent, and of full entropy, so that values of the inner states cannot be exploited to build collisions. Thus, in order to ensure collision resistance, it is sufficient in our case to prevent the existence of high probability differentials of the shape $H(M) + H(M + \delta) = 0$. We then rely on the following assumption to investigate these.

Assumption 1. *The highest probability of a differential trail is a good indication of the highest probability of a differential.*

Thanks to Assumption 1, estimating the security level can be done by modeling the differential propagation with a MILP model and this is now a widespread practice [JN16, SLN+21].

Goal 4. *A lower bound on the number of active S-boxes in the differential trails of a candidate should be easily computed with computer-aided tools, such as MILP solvers.*

Section 5 will be fully-dedicated to our MILP modeling and its optimizations.

2.4 A roadmap to achieve these goals

All those guiding principles lead us toward the family of UHF that we describe in the next section. Our goal are in line with previous works [JN16, SLN+21]: we want a primitive that favors parallel AES calls to optimize scheduling. However, properly taking this into account means carefully considering the number of 128-bit XORs, and in fact minimizing it—the authors of Rocca already observed the negative impact that AES and XOR used "in a cascade way" could have. As a consequence, we limit ourselves to sparse linear layers.

To compensate the slower diffusion implied by the sparse linear layer, and to broaden our search space, we consider more sophisticated injection techniques inspired by the design (tweak-)key schedules. This could increase the cost of each round (in particular in terms of memory), but it indeed enables the safe use of very simple round functions. This overall structure is similar to that of Panama [DC98], a hash function attacked in [RVPV02]. It was based on a large "buffer" and a smaller "inner state", the former being linearly updated using message blocks, and the latter being non-linearly updated using data extracted from the buffer. The separation between buffer and inner state was quickly set aside as several algorithms adopted a similar structure that nevertheless involved a datapath from the inner state to the buffer, e.g. RadioGatùn [BDPA06] and Lux [NBK08].

Our whole construction is presented in the next section.

3 A Specific Family of Universal Hash Functions

In light of the discussion presented in the previous section, we have settled on a specific family of UHFs, that is both large enough to contain algorithms that are both fast and secure, and that is small enough that we can practically explore vast subsets of it.

The idea is to separate the (potentially large) state into two subparts with different roles: an inner part updated with AES rounds and a linear layer, and an outer part updated only with a linear layer and new message blocks. Each round, words of the outer state are XORed to the inner state (but not the other way). The aim is that each message block is XORed several times into the inner state, so that short differential trails leading to collisions do not exist. This construction is similar to many sponge-like constructions, but in our case the linear outer state allows to save many AES round calls (while sponge-like designs will apply the same function to the full state), and to be easily modelable in MILP. This also resembles a large tweakable block cipher with a large tweak, and a linear tweakey schedule. We chose this structure as it has the potential to offer both high throughput (thanks to its reliance on the AES rounds, the expensive operations being restrained to one subpart of the state, the potentially low rate) and high security (thanks to the sparsity of the round which makes it easier to use automated tools to check for differential attacks).

3.1 Notation

Vectors of 128 bits are denoted *word*, *block*, *register* or *wire* depending on the context. The additions are performed bitwise in $\mathbb{F}_2$; they correspond to XORs. The cardinality of a set S is $|S|$. The number of non-zero elements of a vector $v \in \mathbb{F}_2^k$ is denoted $\mathrm{Supp}(v)$. For any $\ell \in \mathbb{N}$, we denote $[\![0, \ell]\!] := \{0, \cdots, \ell\}$. Given a field F, we denote $\mathcal{M}_{u\times v}(F)$ the sets of matrices over F of size $u \times v$. We denote $\mathcal{M}_u(F)$ (resp. $\mathrm{GL}_u(F)$) the set of matrices over F of size $u \times u$ (resp. of invertible matrices over F of size $u \times u$). A diagonal block matrix whose diagonal is made of matrices $A_0, \cdots, A_\ell$ is denoted $\mathrm{Diag}(A_0, \cdots, A_\ell)$. In a block matrix definition, $\star$ denotes an arbitrary block.

3.2 Overall structure

The UHF family we consider is described in Figure 1. Each *wire* on the figure represents a 128-bit value. The inner state is on the left-hand side of Figure 1, and the outer linear message-schedule with memory on the right. Overall, our approach can be seen like a standard Substitution Permutation Network (SPN): the inner state (alternatively denoted X, Y, Z) is iteratively updated through a round function built by composing a linear layer with a non-linear one. Between each round, the linear message-schedule ingests several blocks of the input message, and produces an *injected value* V which is added to Z to yield X. The memory registers of the linear message-schedule, that we denote R, keep linear information on previous input message blocks.

Parameters. From now on, by *size*, we always mean the number of 128-bit blocks. Thus, each member of the family is parameterized by the sizes s, m, r of the inner state X, Y, Z, of the input message M, and of the memory R, that can be chosen freely. Note that s also corresponds to the size of the injected value V. Once these sizes are fixed, we define a specific instance by choosing the vector a and the matrices L, T.

The Boolean vector $a := (a_0, \cdots, a_{s-1})$, of size s, indicates whether a state wire goes through an AES round or not. For any i such that $a_i = 1$, the i-th wire of the state is called an *AES wire*.

The $s \times s$ invertible sparse matrix $L \in \mathcal{M}_{s\times s}(\mathbb{F}_{2^{128}})$ is used as linear layer. By design, we restrict the coefficients of L to $\{0, 1\}$, so that L can be viewed as a matrix of $\mathcal{M}_{s\times s}(\mathbb{F}_2)$.

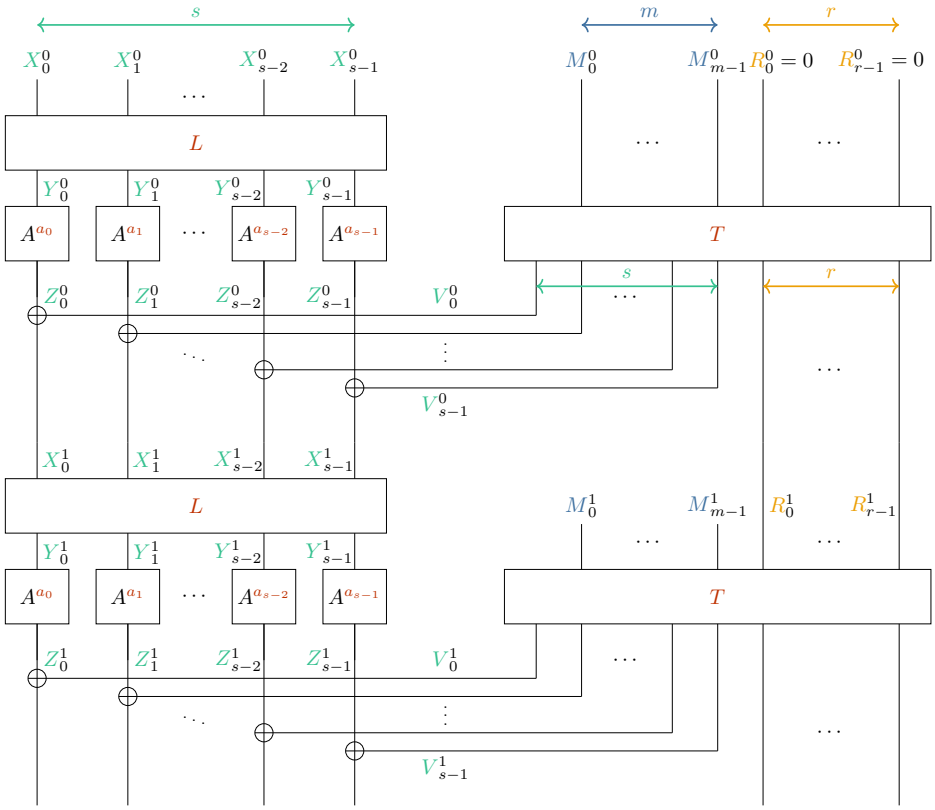

Figure 1: A stands for a key-less AES round. Each choice of the size parameters s, m, r, the Boolean values a_i, and the linear matrices L, T defines an instance of the framework.

In particular, the output of the linear layer is only composed of copies and XORs of the 128-bit input words.

Finally, T is the $(s+r) \times (m+r)$ message-schedule *transition matrix*. T indicates how to compute the s-word injected-value V and how to update the memory R (of size r). Both are linearly computed using the current memory R and m fresh message words, M. Similarly to L, we restrict by design the coefficients of T to $\{0,1\}$: $T \in \mathcal{M}_{(s+r)\times(m+r)}(\mathbb{F}_2)$.

Notation 1 (Time stamp, coordinates and sequences)**.** *As the values of the blocks vary over time, we use superscript to indicate the clock (with $t = 0$ as initial clock) while subscripts are reserved for coordinates: for instance R_i^t stands for the i-th coordinate of R^t, that is, R at time t. We keep plain characters for generic purposes: e.g. the memory R, and use calligraphic letters to denote the sequence throughout time: e.g. $\mathcal{V} := (V^t)_{t\in\mathbb{N}}$. Finally, for any finite subsets $I \subset \mathbb{N}$, $J \subset [\![0, s-1]\!]$ and $t \in \mathbb{N}$, we denote sub-sequences and sub-vectors as: $\mathcal{V}^I := (V^t)_{t\in I}$ and $V_J^t := (V_j^t)_{j\in J}$.*

3.3 Round function and message-schedule

Round Function. It is applied on the inner state, and is composed of three layers:

- a linear layer $Y^t := L(X^t)$.
- an AES-round layer where an AES round A (composed of SubBytes, ShiftRows and MixColumns, but *without* AddKey) is applied in parallel to each AES wire: $Z_i^t := A^{a_i}(Y_i^t)$; where $A^0 := \mathrm{Id}$ and $A^1 := A$.

- and an injected-value addition layer where the injected value V^t of round t, generated by the message-schedule, is added to the state: $X^{t+1} := Z^t + V^t$.

In the AES round layer, the AddKey step is omitted. Thus, by using the AddKey step of the AES-NI instruction, the addition of the round-value word is free on AES wires.

Message-Schedule. The linear message-schedule has a memory R of size r. Each register contains a linear combination of previous message words. At round t, m new message words are ingested, the s-long injected value V^t is output and the memory R^t is updated, in a single transition step:

$$\forall t \geqslant 0, \quad \begin{pmatrix} R^{t+1} \\ V^t \end{pmatrix} = T \begin{pmatrix} R^t \\ M^t \end{pmatrix}. \tag{1}$$

As highlighted by the previous equation, it is convenient to decompose T as a block matrix.

Notation 2 (T decomposition)**.** *In the following, given a transition matrix T, we will intensively use the following decomposition and notation:*

$$T := \begin{pmatrix} \overset{r}{\longleftrightarrow} & \overset{m}{\longleftrightarrow} \\ T_{00} & T_{01} \\ T_{10} & T_{11} \end{pmatrix} \begin{matrix} \updownarrow r \\ \updownarrow s \end{matrix}. \tag{2}$$

Taking advantages of Equations (1) and (2), we can easily express the injected-values as (recursive) linear combinations of input-messages blocks:

$$\forall t \geqslant 0, \quad R^{t+1} = T_{00}R^t + T_{01}M^t \qquad V^t = T_{10}R^t + T_{11}M^t. \tag{3}$$

Remark 1. Let $I \subset \mathbb{N}$, $u := max(i \mid i \in I)$. $\mathcal{V}^I$ can therefore be viewed as a family of $|I|s$ linear combinations, or equivalently as a $|I|s \times um$ matrix where each column represents one of the um message blocks that can appear in the $|I|s$ combinations. We will often prefer the latter point of view.

An injected-value sequence $\mathcal{V}$ can be obtained from infinitely many matrices T. For instance, infinitely many unused memory registers could be added. It is thus necessary to limit as much as possible this redundancy while exploring the transition matrices T. In the next section, we start by finding a "normal form" for the transition matrices we will study. We then limit our search by defining an equivalence relation between injected-value sequences and finally present and justify our search space.

4 A Searchable Space of UHFs

4.1 A normal form for transition matrices

The first notable point about transition matrices is that, at clock t, only the space spanned by the memory registers (and not the register themselves) matters. Indeed, the same information can be recovered from two different spanning families, only in different representation systems. This is illustrated by the following proposition which is proved in Appendix B.

Proposition 1 (Change of basis for memory registers)**.** *Let T be a transition matrix. Let $P \in \mathrm{GL}_r(\mathbb{F}_2)$. Let us define $T_P \in \mathcal{M}_{(s+r)\times(m+r)}(\mathbb{F}_2)$ such that:*

$$T_P = \begin{pmatrix} PT_{00}P^{-1} & PT_{01} \\ T_{10}P^{-1} & T_{11} \end{pmatrix}. \tag{4}$$

Then T_P produces the same sequence $\mathcal{V}$ as the original matrix T.

Sketch of Proof. As P is invertible, the rows of $(T_{00}|T_{01})$ and $(PT_{00}|PT_{01}) = P(T_{00}|T_{01})$ spans the same space. The multiplication of P^{-1} to the right of PT_{00} and T_{10} only adapts the linear operators to the newly-chosen spanning family. □

For fixed sizes r, m, s, Proposition 1 in particular states that it is sufficient to explore a single representative per similarity class for the top-left block T_{00}. We recall the following classical results about similarity that can be for instance found in [DF04, Thm. 14, p. 476] or [Gan90, p. 192].

Definition 3 (Similarity)**.** Let $n \in \mathbb{N} \setminus \{0\}$. Let F be a field. Let $M, N \in \mathcal{M}_n(F)$. M is *similar* to N if there exists a matrix $P \in \mathrm{GL}_n(F)$ such that $M = PNP^{-1}$.

Proposition 2 (Normal Form for similarity)**.** *Let $M \in \mathcal{M}_n(F)$. Let us denote C_Q the companion matrix associated to a polynomial $Q \in F[X]$.*

1. *Similarity is an equivalence relation over $\mathcal{M}_n(F)$. We denote it $\sim$.*
2. *There exists a unique family $(Q_0, \cdots, Q_{\ell-1})$ of polynomials such that:*

$$Q_{\ell-1} \mid \cdots \mid Q_1 \mid Q_0 \quad \textit{and} \quad M \sim \mathrm{Diag}(C_{Q_0}, \cdots, C_{Q_{\ell-1}}).$$

This representative is called Frobenius Normal Form *or* Rational Canonical Form.

According to Proposition 2, it is thus sufficient to exhaust all possible Frobenius Normal Forms rather than all $r \times r$ matrices for the top left-hand corner. This decreases the search space by a significant factor: for $r = 4$, there are $20160 \approx 2^{14.3}$ matrices in $\mathrm{GL}_4(\mathbb{F}_2)$, but only $14 \approx 2^4$ equivalence classes[2].

On top of that, Proposition 1 also allows to get rid of redundant memory registers, as presented by the following corollary, proved in Appendix B.

Corollary 1. *Let T be a transition matrix. Let us denote $d = \mathsf{rank}(T_{00}|T_{01})$. Then, there exists an instance using d memory-registers which generates the same sequence $\mathcal{V}$.*

Sketch of Proof. If $(T_{00}|T_{01})$ is not full-ranked, that is $d < r$, then the memory registers are not independent. An extracted basis (of size d) is thus enough to express any linear combination of memory registers with strictly less memory. □

Corollary 1 states that after choosing a Frobenius Normal Form for T_{00}, and any value for T_{01}, one can immediately look at the rank of the top half $(T_{00}|T_{01})$. If the top half has not full rank, the study of the matrix comes back to the study of an instance with strictly less memory (a smaller r). If the search is done by increasing values of r, one can only consider a top half with a full rank.

4.2 An equivalence relation for injected-value sequences

Even if we limit redundancies thanks to Proposition 1 and Corollary 1, for most of values of r, m, s, the associated space of message-schedules remains too big. In particular, it cannot be exhaustively searched, especially if a MILP problem needs to be optimized *for each instance*.

To further reduce the explored space, we first restrict ourselves to matrices T for which $\mathsf{rank}(T_{11}) = m$. Indeed, if $\mathsf{rank}(T_{11}) < m$, only a strict subspace of the messages at round t impacts the injected values at this round. This does not directly generate collisions, since the unused messages can be stored in memory and used in later rounds. However,

[2] Counting the number of equivalence classes is easier using another normal form for similarity. Any invertible matrix is similar to one of the form $\mathrm{Diag}(P_0^{d_0}, \cdots, P_{\ell-1}^{d_{\ell-1}})$ where each P_i is irreducible [Gan90, Thm 12', p. 199]. When $\ell = 4$, there exist 14 such families of polynomials.

this requires extra registers whose only purpose is to store the unused injected messages of previous rounds, increasing the memory size r without increasing the security. More precisely, after a few rounds, such an instance behaves as if exactly m message blocks impacted the injected values at each round; the message blocks sequence being slightly slid. So from now on, $\mathsf{rank}(T_{11}) = m$, and in particular, $s \geqslant m$.

Secondly, we take into account our adversary in a scenario where it has a full control over the input differences in message blocks (such as a chosen-plaintext scenario). From this point-of-view, the implementation does not matter, only the *actual* decompositions of all V_i^t as linear combinations of $M_j^{t'}, i, j \in [\![0, m-1]\!], t' \leqslant t$ do. In particular, with n degrees of freedom, such an adversary can choose the differences of n independent V_i^t, rather than just the differences of n message blocks M_i^t. We thus study injected-value sequences up to linear change of variables of the inputs.

Definition 4 (Linearly-equivalent injected-values sequences.)**.** Let $\mathcal{V} = (V^t)_{t\in\mathbb{N}}$ and $\mathcal{W} = (W^t)_{t\in\mathbb{N}}$ be sequences of linear combinations such that, for any i, t, V_i^t depends only on $M_j^{t'}$, where $j \in [\![0, m-1]\!], t' \leqslant t \in \mathbb{N}$. Then, $\mathcal{V}$ is *linearly-equivalent* to $\mathcal{W}$ if:

$$\forall\, t \in \mathbb{N} \setminus \{0\},\ \exists\ P^t \in \mathrm{GL}_{tm}(\mathbb{F}_2), \quad V^{[\![0,t-1]\!]} = W^{[\![0,t-1]\!]} P^t;$$

where P^t is a *lower triangular block matrix*[3] whose blocks are of size $m \times m$.

Remark 2. Let $t \in \mathbb{N} \setminus \{0\}$. The lower triangular form of P^t implies that the equivalence relation preserves the fact that only variables $M_i^{t'}$, $t' \leqslant t$ appear in both V^t and W^t.

Proposition 3. *Linear equivalence of injected-value sequences, as defined in Definition 4, is an equivalence relation.*

Sketch of Proof. The invertible lower triangular matrices is a sub-group of $\mathrm{GL}_{tm}(\mathbb{F}_2)$. Let $t > 0$ and $V^{[\![0,t-1]\!]} = W^{[\![0,t-1]\!]} P^t$, $W^{[\![0,t-1]\!]} = X^{[\![0,t-1]\!]} Q^t$. Reflexivity is proved with Id, symmetry using $(P^t)^{-1}$ and transitivity using $P^t Q^t$. □

Proposition 4. *[Proved in Appendix B.] Let T be a transition matrix such that $\mathsf{rank}(T_{11}) = m$. Then, up to a wire permutation of the inner state, T produces a sequence $\mathcal{V}$ which is linearly-equivalent to the sequence produced by $\widetilde{T}$, where:*

$$\widetilde{T} = \begin{array}{c} \overset{r}{\longleftrightarrow}\ \overset{m}{\longleftrightarrow} \\ \left(\begin{array}{cc} \star & \star \\ \hline 0 & \mathrm{Id}_m \\ \star & \star \end{array}\right) \begin{array}{l} \updownarrow r \\ \updownarrow m \\ \updownarrow s-m \end{array} \end{array}.$$

Sketch of Proof. We introduce the following decompositions:

$$\begin{pmatrix} T_{10} & T_{11} \end{pmatrix} = \begin{array}{c} \overset{r}{\longleftrightarrow}\ \overset{m}{\longleftrightarrow} \\ \begin{pmatrix} B & C \\ D & E \end{pmatrix} \begin{array}{l} \updownarrow m \\ \updownarrow s-m \end{array} \end{array} \quad \text{and} \quad \widetilde{T} := \left(\begin{array}{c|c} T_{00} - T_{01}C^{-1}B & T_{01}C^{-1} \\ \hline 0 & \mathrm{Id}_m \\ D - EC^{-1}B & EC^{-1} \end{array}\right). \quad (5)$$

Because $\mathsf{rank}(T_{11}) = m$, up to a wire permutation of the inner state, we can suppose that $C \in \mathrm{GL}_m(\mathbb{F}_2)$. With the following lower-triangular change of variables:

$$\forall t \in [\![0, \ell-1]\!], \quad \widetilde{M}^t := BR^t + CM^t \iff C^{-1}(\widetilde{M}^t - BR^t) = M^t;$$

[3] $V^{[\![0,t-1]\!]}$ and $W^{[\![0,t-1]\!]}$ are viewed as matrices of dimension $ts \times tm$, see Remark 1.

we can rewrite Equation (3) as:

$$R^0 = 0, \quad \forall t \geqslant 0,\ R^{t+1} = (T_{00} - T_{01}C^{-1}B)R^t + T_{01}C^{-1}\widetilde{M}^t,$$
$$V^t_{[\![0,m-1]\!]} = \widetilde{M}^t, \quad V^t_{[\![m,s-1]\!]} = (D - EC^{-1}B)R^t + EC^{-1}\widetilde{M}^t.$$

This implies that the sequence $\mathcal{V}$, is linearly-equivalent to the sequence generated by the transition matrix $\widetilde{T}$. □

We can now present the chosen form for the explored transition matrices.

Theorem 1. *Let T be a transition matrix such that $\mathsf{rank}(T_{11}) = m$. Then, up to a wire permutation of the inner state, T produces a sequence $\mathcal{V}$ which is linearly-equivalent to the sequence produced by a matrix $\widetilde{T}$ of the following form:*

$$\widetilde{T} = \left(\begin{array}{cc} F & \star \\ \hline 0 & \mathrm{Id}_m \\ \star & \star \end{array}\right) \begin{array}{l} \updownarrow r \\ \updownarrow m \\ \updownarrow s-m \end{array} \quad ; \tag{6}$$

where F is a Frobenius Normal Form matrix.

Proof. First using Proposition 4, we obtain, up to a wire permutation of the inner state, a transition matrix $\widetilde{T}$ which produces a linearly-equivalent sequence, as in Equation (12). We can now use Proposition 1, in order to put the top left-hand block in Frobenius Normal Form. The multiplication of the lower-left part by P^{-1} does not change the fact that the first row of this block are all-0. The lower-right block is not modified, so Id_m still appears on its first rows. T' has thus the announced form. □

The class of matrices presented in Theorem 1 is not only chosen to make the search more efficient, but also for its *sparsity* to guarantee a small implementation cost. Indeed, the Frobenius Normal Form constitutes a very sparse representative of a similarity class: it is a sparse matrix (a diagonal block matrix) with sparse non-empty blocks (companion blocks). The chosen form for the lower half is also quite sparse with the 0 and Id blocks.

4.3 Constraints on the linear layer

Regarding the linear diffusion matrix L, it should be implementable with a low number of XORs. However, we must ensure that each inner state block at round i will eventually influence all of them. To this end, we use the following metric of diffusion.

Definition 5. Let $\hat{L}$ be a matrix identical to a binary matrix L, except that its coefficients are integers. The *diffusion time* of L is the smallest integer i such that all coefficients in $(\hat{L})^i$ are non-zero. If no such integer exists, we set it to $+\infty$.

We consider integers rather than binary field elements so that additions do not cancel out; this is equivalent to considering the iterations of L, such that all XORs in the matrix multiplications are replaced with ORs. Intuitively, this number tells how many rounds are needed to ensure a full diffusion in the inner part, although in some special cases, it is not entirely accurate as there may be some bad interactions between non-AES wires and the linear layer L. In the case where all wires are AES wires, this metric is exactly the number of rounds which guarantee that every output wire depends on every input wire. In our search space, we generate matrices L under weight constraints, often with a weight of $s+1$ or $s+2$ so that L can be implemented with 1 or 2 XORs and ignore matrices with high diffusion time: we mostly use a value of around $2 \times s$ in this paper[4].

[4]For a rate of 1.75, we show candidates with infinite diffusion time.

4.4 The actual explored space

The search method presented above is optimized but heuristic : we stress that we do not assure the minimal sparsity of the studied transition matrices. Still, the explored space contains promising candidates (see Section 6), that could be further-optimized later on. Nevertheless, exhaustive search remains unreachable. Equivalence relations on a and L could be used, but would (and in practice do) interfere with the previous ones. Instead, we restrict the weight of a and L, as described in Section 4.3 and further in Section 6.

5 Turning Collision Resistance into a MILP Problem

The search space being established, we now focus on assessing the security of the potential UHF candidates, by building an adapted MILP model and then solving it thanks to an optimizer. A MILP model is composed of three objects: *variables*, representing either real numbers or (modular) integers[5], *constraints*, that is, inequalities between $\mathbb{Z}$-affine combinations of variables, and an *objective function* which is a $\mathbb{Z}$-linear combination of variables that need to be maximized (or minimized) when subjected to the given constraints. A MILP solver, such as Gurobi [Gur23], takes as input a MILP model and returns, if it exists, *values* for the variables that both satisfy the constraints and maximizes (or minimizes) the objective function.

5.1 Prior works

The use of MILP modeling for searching differential trails with the highest probability was set to light by Mouha, Wang, Gu & Preneel in 2011 [MWGP12]. Several approaches exist depending on the needed level of precision and the available computational power. In theory, by using one MILP variable for each bit of the state at each round, all the non-linear differential transitions could be modeled (at the cost of *many* constraints). This approach is in practice very costly. For byte-aligned (resp. nibble-aligned) primitives, it is much faster and practical to rather affect a MILP variable to each nibble (resp. byte) of the state. Yet less precise, such a model enables (if it can be efficiently solved) to determine the minimum number of active S-boxes, from which an upper bound on the probability of the best differential trail can easily be estimated. In the case of AES-based ciphers, this method has become standard, as highlighted by Rocca [SLN+21] or Deoxys-BC [JNPS21]. Following their lead, we consider the byte-wise approach.

To do so, we extend Notation 1 so that the byte position appears.

Notation 3. *The second subscript indicates the byte position:* $X^i_{j,\ell}$ *is the* ℓ*-th byte of* X^i_j.

5.2 Our model

From now on, a candidate has been chosen: s, m, r and a, T, L are now fixed. To these constants, we add ρ, the number of rounds of the primitive to model.

Variables. Let $i \in [\![0, \rho-1]\!]$ be a round number, $j \in [\![0, B-1]\!]$ be a word number (where the bound $B \in \{s, m, r\}$ depends on the register we look at) and $\ell \in [\![0, 15]\!]$ be a byte position. We track the differential activeness of every byte throughout the rounds by modeling each byte of the state as a *binary variable*, that is equal to 0 if the byte is inactive and 1 if it is active. We use lowercase to denote the binary variables. More precisely $x^i_{j,\ell}, y^i_{j,\ell}, z^i_{j,\ell}, r^i_{j,\ell}, m^i_{j,\ell}, v^i_{j,\ell}$ respectively represents the bytes $X^i_{j,\ell}, Y^i_{j,\ell}, Z^i_{j,\ell}, R^{j,\ell}_i, M^{j,\ell}_i, V^i_{j,\ell}$.

[5] "Mixed" in MILP actually highlights the different natures of variables.

Objective. Our goal is to minimize the number of active S-boxes, represented by the variables $y^i_{j,l}$ on AES wires (*i.e.* $j \in \mathrm{Supp}(a)$):

$$\mathrm{Obj} := \sum_{i=0}^{\rho-1} \sum_{j \in \mathrm{Supp}(a)} \sum_{\ell=0}^{15} y^i_{j,l}.$$

Now, we present constraints that will appear in the definition of more advanced ones.

Multiple-XOR. It models the relation $\bigoplus_{i=0}^{N-1} U_i = 0$ where $(U_i)_{i \in [\![0,N-1]\!]}$ is a list of N bytes represented by N binary variables $(u_i)_{i \in [\![0,N-1]\!]}$. To do so, we introduce an auxiliary binary variable α, and two constraints:

$$\alpha N \geqslant \sum_{i=0}^{N-1} u_i \quad \text{and} \quad \sum_{i=0}^{N-1} u_i \geqslant 2\alpha.$$

In that way, depending on $\alpha \in \{0,1\}$, either 0 or at least 2 bytes are active.

MDS constraint. It models the relation between an input column of bytes, represented as the binary variables $(y_i)_{i \in [\![0,3]\!]}$, and an output one, represented as $(z_i)_{i \in [\![0,3]\!]} \in \{0,1\}^4$, through the AES MDS matrix. With an auxiliary binary variable α, and the two constraints:

$$10\alpha \geqslant \sum_{i=0}^{3} y_i + z_i \quad \text{and} \quad \sum_{i=0}^{3} y_i + z_i \geqslant 5\alpha;$$

either 0 or at least 5 bytes are active.

Remark 3. In the above constraints, Σ corresponds to an integer sum, *not a modulo-2 sum.*

We can now create constraints for each layer of the round function. Let $i \in [\![0, \rho-1]\!]$.

Linear layer. The transition through L is naturally expressed by linear relations between bytes. Denoting $L = (L_{j,k})_{j,k \in [\![0,\cdots,s-1]\!]}$ (where $L_{j,k} \in \mathbb{F}_2$ for any j,k), it holds that:

$$\forall i \in [\![0,\rho-1]\!], j \in [\![0,s-1]\!], \ell \in [\![0,15]\!], \qquad Y^i_{j,\ell} = \sum_{k=0}^{s-1} L_{j,k} X^i_{k,\ell}.$$

These constraints can therefore be modeled using a Multiple-XOR constraint.

AES-round layer. Let $j \in \mathrm{Supp}(a)$ so that an AES round is applied on the j-th wire. The S-box layer does not change the activity pattern, but the linear layer (ShiftRows and MixColumns) needs to be modeled. For any round $i \in [\![0,\rho-1]\!]$, and column index $t \in [\![0,3]\!]$, the t-th diagonal of Y^i_j is linked by a MDS relation together with the t-th column Z^i_j. Those relations require a MDS constraint.

When $j \notin \mathrm{Supp}(a)$, we simply add the constraints $y^i_{j,\ell} = z^i_{j,\ell}$ for all i, ℓ.

Message-schedule. The 128-bit linear relations between R^i, M^i, R^{i+1}, V^t given by Equation (3) can be modeled with 16 Multiple-XOR constraints (one for each byte).

Injected-value addition. For all i,j,ℓ, $Y^{i+1}_{j,\ell} = Z^i_{j,\ell} + V^i_{j,\ell}$ is modeled as a Multiple-XOR.

Finally, we add constraints on the inputs/outputs of the UHF, and constraints to take advantage of the inherent symmetries of the AES round function.

Input constraints. At clock $t = 0$, the state and memory are fully inactive. Thus,

$$\forall \ell \in [\![0, 15]\!], j \in [\![0, s-1]\!], j' \in [\![0, r-1]\!] \qquad x^0_{j,\ell} = 0, \quad r^0_{j',\ell} = 0.$$

Message constraints. If a trail with an inactive first round exists, shifting it by 1 round makes it still a valid trail. Moreover, in the AES, any column (resp. row) plays the same role, so any trail can be shifted so that the first difference appears in the byte of index $\ell = 0$. By forcing at least one 0-index first-round-message byte to be active, we facilitate the solving process, without leaving any trail aside. Hence the *symmetry constraint*:

$$\sum_{j=0}^{m-1} m^0_{j,0} \geqslant 1.$$

This model will be referred as our *basic model*. Additionally, we can add to this model some output and/or linear incompatibilities constraints.

Output constraints. We can force the state to be fully inactive at the end:

$$\forall \ell \in [\![0, 15]\!], j \in [\![0, s-1]\!] \qquad x^\rho_{j,\ell} = 0.$$

This constraint highly reduces the MILP solution space. However it is a too-strong constraint when ρ is small: a differential trail over more rounds but with less active S-boxes cannot be captured by the model. In practice, we iteratively increase ρ to capture more and more trails, until a sufficient number of rounds is reached.

Removing linear incompatibilities. With the basic model, some obtained activity patterns may not be instantiable into differential trails because of linear incompatibilities, similar to the ones observed on AES [FJP13] or on Deoxys-BC [CHP+17]. Unlike those ciphers, our message-schedule is acting on 128-bit words which enables us to model the linear incompatibilities with exact constraints[6]. In our case, for an AES wire of index j, we observe that $\mathsf{MC} \circ \mathsf{SR} \circ \mathsf{SB}(Y^i_j) \oplus V^i_j = X^{i+1}_j$. Introducing, the variables

$$\widehat{X}^i_j := \mathsf{LIN}^{-1}(X^i_j), \quad \widehat{V}^i_j := \mathsf{LIN}^{-1}(V^i_j), \quad \widehat{M}^i_j = \mathsf{LIN}^{-1}(M^i_j), \quad \widehat{R}^i_j = \mathsf{LIN}^{-1}(R^i_j), \quad (7)$$

where $\mathsf{LIN} := \mathsf{MC} \circ \mathsf{SR}$, we can rewrite it as:

$$\forall j \in \mathrm{Supp}(a), i \in [\![0, \rho-1]\!] \qquad \mathsf{SB}(Y^i_j) \oplus \widehat{V}^i_j = \widehat{X}^{t+1}_j. \quad (8)$$

Getting rid of linear incompatibilities in the model precisely means taking Equation (8) into account. To do so, we introduce the binary MILP variables $\hat{x}^i_{j,\ell}, \hat{v}^i_{j,\ell}, \hat{m}^i_{j,\ell}, \hat{r}^i_{j,\ell}$ corresponding to the bytes of $\widehat{X}^i_j, \widehat{M}^i_j, \widehat{R}^i_j, \widehat{V}^i_j$. We then build constraints corresponding to Equation (7) using MDS-constraints (SR only consists in a renumbering). The bytes of $\widehat{R}^i_j, \widehat{M}^i_j, \widehat{R}^{i+1}_j$ and $\widehat{V}^i_j$ are linearly-dependent with respect to Equation (3); this is encoded using multiple-XOR constraints. Finally, because S-box$(Y^i_{j,\ell})$ is active if and only if $Y^i_{j,\ell}$ is, Equation (8) is encoded byte-wise with 3-XORs between $y^i_{j,\ell}$ (no hat), $\hat{v}^i_{j,\ell}$, and $\hat{x}^i_{j,\ell}$.

In practice the model solving is severely slowed by taking these incompatibilities into account. It however often increases the minimal number of active S-boxes by a few. Because of the pros and cons of each of these additional constraints, we parameterize our model depending on them. In Section 6, we explain how we parameterized the models to converge toward promising candidates.

[6] On the contrary, the tweakey-schedule of Deoxys-BC includes a byte permutation. This is the reason why, Cid *et al.* used heuristic constraints based on degrees of freedom.

Notation 4. *Let $\rho \geqslant 1$, and* lin *and* output *be two Booleans. We denote by* Model(ρ, lin, output) *the model corresponding to ρ rounds, where the linear incompatibilities contraints (resp. output constraints) are considered if* lin = True *(resp.* output = True*) and not otherwise.*

5.3 A word on solutions

As already mentioned, a solution to these models consists in an activity pattern, which, *if it is instantiable*, minimizes the number of active S-boxes. There is however no *a priori* guarantee that it actually can be instantiated as an actual differential trail. Nevertheless, if it is instantiable, and if all transitions can occur with maximal probability, then the instantiated trail would have a probability of p^N, where N is the number of active S-boxes, and $p = \delta 2^{-k}$ is the probability associated to the differential uniformity δ of the k-bit S-box. Thanks to Assumption 1, this higher bound on the probability of the best differential trail enables to estimate the level of security of any candidate (once the solver terminates). Section 6 presents our experimental results.

6 Experimental Results of the Search for Good Candidates

6.1 Search strategy

In order to find good candidates, we proceed as follows.

1. First, we fix some numerical values for s, m, r, and $w := |\mathrm{Supp}(a)|$, a maximum number of XORs to implement L and T, and a diffusion time threshold for L.
2. Then, we generate random candidates for a, L, and for T according to Section 4.
3. For each of them, we solve Model(ρ, lin=False, output=True) using Gurobi [Gur23], with increasing ρ.
4. For each candidate with a sufficient number of active S-boxes (*i.e.* more than 24), we generate assembly code corresponding to the round function, and benchmark it on a recent CPU. If the software performance is high enough, we keep the candidate.
5. At last, we select one of the final candidates based on performance/security trade-off, and perform a last MILP solve of Model(ρ, lin=True, output=False) with high ρ to guarantee the security of the candidate.

In Step 2, in practice, a is generated randomly among vectors of s elements with hamming weight m, L is generated from a random element of the symmetric group $S_s(\mathbb{F}_2)$, of which k 0s are replaced by 1s (the implementation of L thus requires k XORs at most), and T is generated by looping over the set of possible Frobenius Normal Forms, until the XOR-cost is less than j. The rest of the matrix T is generated, line by line, by making sure that the XOR-cost constraint is satisfied. In our search, k and j are empirically randomized.

In Step 3, by making ρ bigger, we go from very restrictive and quickly-solved models to more complete but slower ones. Optionally, Step 4 can be executed before increasing ρ, to discard non-performing candidates and avoid time-consuming MILP-solves.

Running the Search. In practice, we select $\rho \in \{2, 3, 4, 12, 20\}$. At each point, if the number of active S-boxes falls under a *security threshold*, the candidate is discarded. The security threshold is fixed to 20 active S-boxes, but by using lin=True in a later solve, the minimal number of active S-boxes might increase. Between the runs with $\rho = 12$ and $\rho = 20$, we automatically generate a C implementation, compile it on-the-fly and

benchmark it. If the speed (in cycles per byte) of the candidate, falls under a *speed threshold*, the candidate is discarded. This threshold depends on the parameters (chosen in Step 1) and of the processor used in the benchmark.

Because our candidates rely on AES-NI instructions, their speed is upper-bounded by $u \times v$ cycles per 128-bit, where u is the throughput of AESENC, and v the rate (see Section 2.1). Thus, candidates with a speed close to this bound are considered promising. In our case, we benchmark on an Intel 11th Gen Core i5-1135G7 (Tiger Lake family), with a throughput $u = 0.5$, and we mainly target round functions with rate $v = 2$. Those round functions cannot go faster than 1 cycle per 128-bit state, *i.e.* 0.0625 cycles per byte. For rate-2 candidates, if $w = 8$, we set the speed threshold to 0.08 cycles per byte; if $w < 8$, very few candidates are faster than 0.08 cycles per bytes, so the threshold is increased accordingly.

For each remaining candidate, we finally solve Model(20, lin=True, output=False) in order to obtain a final bound on the number of active S-boxes.[7] This heuristic finds candidates with both good performance and security but may not be the fastest approach. Still, it is much faster than simpler approaches such as solving the slow-but-accurate Model(high ρ, lin=True, output=False) directly, or benchmarking every candidate before running the fastest Model(low ρ, lin=False, output=True).

Choosing the numerical parameters. We chose numerical parameters, based on the following experimental observations. First, for a fixed rate, increasing w (and therefore m) tends to improve the performance. Moreover, when other parameters are fixed, increasing r or s tends to increase the security. Finally, we limited the sizes of the state to $r + s < 16$.

We thus looked for candidates with a high w and multiple memory registers. We also explored lighter candidates, and propose good candidates for smaller values of w, typically $w \in \{2, 4, 6\}$, which are not as fast but might be parallelizable in some scenarios. Finally, for $w = 1$, $s = 1$, $m = 1$, we looked at a rate-2 construction that lies slightly outside the scope of our family by replacing any message block M_0^t with odd t by 0.

6.2 Results of the search

The results of our search are given in Table 3. For each set of numerical parameters, we give the total number of candidates we considered ("Total"), the number among them that satisfied the security threshold ("After Sec."), and the number of among those that also satisfied the speed threshold ("After Speed"). As we can see, the vast majority of the candidates do not satisfy our demanding criteria, but a broad-enough search allows us to find promising candidates. The case of $w = 1$ is peculiar as such candidates are inherently slower (they are not parallelizable), which is why the bottom right cell of the table is left empty. The properties of the most promising candidates are given in Table 2. Interestingly, for a fixed rate, a higher weight w usually means a higher speed.

Although we did not perform a dedicated search to reduce the number of registers used in rate-2 candidates, we note that PetitMac's round function (Figure 3) requires 6 registers in total, and can be implemented without any additional temporary register[8]. Therefore, this improves on the result of Nikolic [Nik17b, Appendix C], which does not find a rate-2 candidate with less than 8 registers; the candidates found may moreover require additional temporary registers in the implementation (see [TSI23] for similar concerns on Rocca).

[7] When output=False, reducing ρ increases the solution space. Thus, a final solve with a smaller ρ would also give a legitimate, but potentially loose lower bound, and may thus decrease the solving time.

[8] Indeed, it is possible to unroll the round function 10 times (5 iterations of Figure 3) to remove the register shifts in the message schedule.

Table 2: Table of the retained candidates over different parameters sets. Speeds were measured on a Intel 11th Gen Core i5-1135G7 (Tiger Lake) for different sizes of message.

								Speed (cy/B)		
Rate	w	m	s	r	XOR-cost	Diffusion	Security	16 kB	256 kB	Descr.
2	8	4	9	4	4	15	26	0.074	0.067	Figure 2
1.75	7	4	10	5	5	∞	23	0.079	0.068	App. C.1
2	6	3	7	4	4	11	25	0.086	0.080	App. C.2
2	4	2	6	4	3	9	24	0.104	0.099	App. C.3
2	2	1	4	3	4	5	23	0.180	0.175	App. C.4
2	1	0.5[1]	1	5	3/1[2]	-	26	0.374	0.371	Figure 3

[1]A message is added every other round.
[2]There is 1 inherent XOR in the transition matrix. Every other rounds, the message accounts for 2 additional XORS.

Table 3: Number of tested and passed candidates for different settings. Candidates were generated so that they satisfy the diffusion threshold. The search time is given in core hours. The search was performed on Cascade Lake Intel Xeon 5218.

Meta parameters			Thresholds				Candidates		
Rate	w	m	Diff. Time	Security	Speed	Search time	Total	After Sec.	After Speed
2	8	4	15-22	22-24	0.07	>300k	8.0M	346	13
2	6	3	12-16	23	0.09	28k	1.2M	154	4
2	4	2	10-12	23	0.105	18k	2.2M	187	3
2	2	1	4-8	23	0.19	30k	1.2M	1,165	13
2	1	0.5	2	21/23[1]	-	1,152	908[2]	9	-

[1]We first used lin=False with a security threshold of 21, then lin=True and a threshold of 23.
[2]We exhausted all candidates with $r \leqslant 5$ up to tweakey sequence equivalence.

7 Concrete MAC Instances

Using the results of our search, we propose two new AES-based MACs: LeMac and PetitMac, both with 128-bit key, nonce and tag. We first present the common framework we use to turn our UHF round functions first into full-fledged UHFs, and second into MACs (Section 7.1). We then provide more detailed benchmarks and compare them with the state of the art in Section 7.2[9].

7.1 Specifications

To turn our fast universal hash function into a MAC, we use the following strategy:

1. Using one of the round functions we obtained in our search, we get an ε-AU family H taking an arbitrary message as input with a $128 \cdot s$-bit output. The family is indexed by the secret initial state, and we conjecture that it is a 2^{-128}-AU family based on our MILP analysis.

2. We compose H with an ε-AXU family C taking a $128 \cdot s$-bit input with a 128-bit output. We obtain an ε'-AXU family $C \circ H$.

3. We use the EWCDM construction [CS16], with the ε'-AXU family $C \circ H$, and the AES block cipher.

[9]The implementations of LeMac and PetitMac can be found on GitHub (https://github.com/AugustinBariant/Implementations_LeMac_PetitMac).

We thus obtain a MAC whose security relies only on the PRF security of AES and the ε-AU security of H, the former being a standard assumption, and the latter being a consequence of our MILP-based analysis.

ε-AXU family C. We build the family C using the sum hashing construction from [CW77, Proposition 8]. Given two ε-AXU family $H_1 : A_1 \to B$ and $H_2 : A_2 \to B$, this construction yields an ε-AXU family $G : A_1 \times A_2 \to B$ defined as $G = \{m \mapsto (h_1(m) \oplus h_2(m)) : h_1 \in H_1, h_2 \in H_2\}$. Concretely, we take the AES block cipher as an ε-AXU family (the ε-AXU security of AES is a consequence of its the security as a PRF), and define the family C as:

$$C : x_0, x_1, \ldots, x_{s-1} \mapsto \bigoplus_{i=0}^{s-1} \mathrm{AES}_{k_i}(x_i)$$

where each AES is keyed independently.

C is a 2^{-128}-AXU family assuming that the AES is a secure PRF, and the composition of the 2^{-128}-AU family H and the 2^{-128}-AXU family C yields a 2^{-127}-AXU family $C \circ H$ using the composition result from [Sti92, Theorem 5.6].

EWCDM. The MAC itself follows the EWCDM construction by Cogliati and Seurin [CS16]:

$$\mathrm{EWCDM}[H, E]_{k_1,k_2,k_3}(M, N) = E_{k_3}\big(H_{k_1}(M) \oplus E_{k_2}(N) \oplus N\big).$$

This construction uses a nonce to obtain high security, but it still provides security up to $2^{n/2}$ queries if the nonces are repeated (or omitted).

When used with unique nonces, EWCDM was initially proved secure up to $2^{2n/3}$ queries, but a more recent result proved security up to essentially 2^n queries: assuming that the adversary makes less than $2^n/67$ queries, Mennink and Neves [MN17] proved that:

$$\mathbf{Adv}^{\mathrm{MAC}}_{\mathrm{EWCDM}[H,E]} \leq \mathbf{Adv}^{\mathrm{PRP}}_{F} + \frac{q}{2^n} + \frac{q^2\varepsilon}{2^n} + 2^{-n}.$$

We use the EWCDM construction because it provides significantly higher security than the more common Wegman-Carter-Shoup construction

$$\mathrm{WCS}[H, E]_{k_1,k_2}(M, N) = H_{k_1}(M) \oplus E_{k_2}(N).$$

Indeed, Wegman-Carter-Shoup only provides $2^{n/2}$ security with unique nonces, and fails completely when nonces are repeated.

Initialization. While the family is indexed by the secret initial state, we suggest to derive it as follows: the branch with index i is initialized to $E_{\mathsf{Kinit}}(i)$, where Kinit is 128-bit secret key, and E is the AES-128 block cipher.

LeMac. It is our ultra-fast MAC algorithm. It takes as input a 128-bit nonce and a 128-bit key, and returns 128-bit digest. It is based on the round function summarized in Figure 2, which corresponds to the fastest promising candidate we found for $w = 8$. A detailed algorithm is provided in Appendix D.

PetitMac. For cases where the high parallel potential of LeMac might not be an advantage (e.g. on smaller processors), we propose instead PetitMac, which is based on the promising candidate we found for $w = 1$ (see Table 2, and Figure 3 for its round function); it has a rate of 2, and ensures the activation of at least 26 S-boxes during absorption. PetitMac takes as input a 128-bit nonce and a 128-bit key, and returns 128-bit digest.

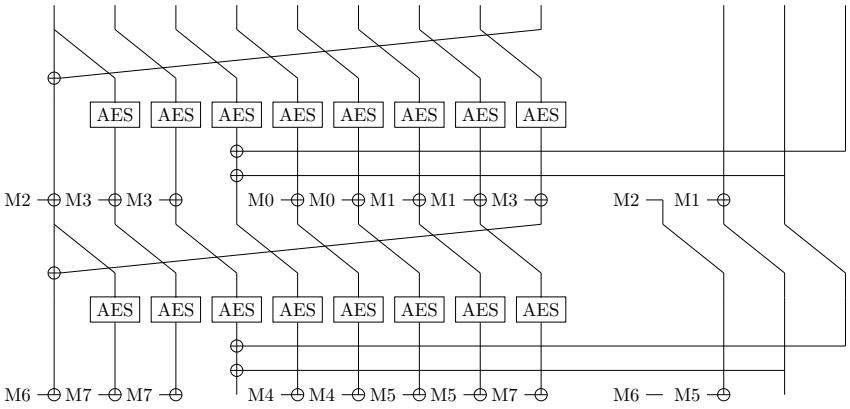

Figure 2: Two rounds of the UHF used in LeMac. For more iterations of the round function, refer to Figure 4 in Appendix D.

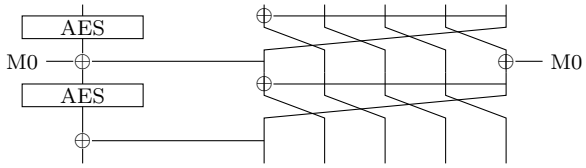

Figure 3: Processing one message block in the UHF used in PetitMac.

Security. We claim that LeMac and PetitMac offer 128-bit security in the nonce-respecting model, meaning that an attacker with advantage close to one requires a data complexity close to 2^{128}, or a time complexity close to 2^{128}. In the nonce-misuse setting, we claim that an attacker with advantage close to one requires a data complexity close to 2^{64}, or a time complexity close to 2^{128}.

7.2 Benchmarks

In order to evaluate the performance of LeMac, we performed comparative benchmarks on several recent hardware architectures from Intel and AMD. We compare LeMac with the following constructions: Rocca [SLN+21] and Rocca-S [NFI24]; AEGIS128 [WP14] and AEGIS128L [WP13]; Tiaoxin-346 v2 [Nik14]; The rate-2 round function of Jean and Nikolić [JN16], with the same initialisation and finalization as LeMac. Rocca, Rocca-S, AEGIS128, AEGIS128L and Tiaoxin-346 are authenticated encryption algorithms, therefore they provide more features than LeMac, but we believe they still provide a reasonable comparison point when used in their associated data-processing mode.

For the benchmarks, we use a hardware performance counter to measure the number of cycles for the execution of the primitive, with various message lengths. All MACs are compiled with `gcc 12.2.0`, and we run the code multiple times, in order to measure performance when the data and code are loaded in the cache. We use the `perf` program to set up a performance counter for the number of elapsed cycles[10], and the `rdpmc` instruction to read the performance counter with low overhead. On Intel CPUs, we obtain the same results using the `rdtscp` instruction, but on AMD CPUs the counter read by `rdtscp` (or `rdtsc`) is independent of the core frequency and does not actually count CPU cycles.

The results are shown in Table 4. We observe that LeMac essentially reaches the maximal possible performance for a rate-2 scheme on these CPU architectures: the Haswell and Skylake architectures compute at most 1 AES round per cycle, corresponding a limit of

[10] Using `perf stat -e cycles:u`

Table 4: Benchmark results.

CPU	Cipher	Speed (c/B)		
		1kB	16kB	256kB
Intel Haswell (Xeon E5-2630 v3)	GCM (AD only)	1.138	0.700	0.605
	Rocca (AD only)	0.602	0.225	0.201
	Rocca-S (AD only)	0.660	0.290	0.269
	AEGIS128 (AD only)	0.809	0.578	0.564
	AEGIS128L (AD only)	0.542	0.299	0.285
	Tiaoxin-346 v2 (AD only)	0.489	0.207	0.190
	Jean-Nikolić	0.455	0.149	0.159
	LeMac	0.498	0.148	0.131
	PetitMac	1.116	0.890	0.876
Intel Skylake (Xeon Gold 6130)	GCM (AD only)	0.817	0.396	0.370
	Rocca (AD only)	0.573	0.190	0.167
	Rocca-S (AD only)	0.568	0.213	0.192
	AEGIS128 (AD only)	0.682	0.470	0.460
	AEGIS128L (AD only)	0.505	0.267	0.253
	Tiaoxin-346 v2 (AD only)	0.473	0.206	0.189
	Jean-Nikolić	0.389	0.142	0.130
	LeMac	0.422	0.144	0.126
	PetitMac	0.792	0.635	0.626
Intel Ice Lake (Xeon Gold 5320)	GCM (AD only)	0.699	0.311	0.286
	Rocca (AD only)	0.528	0.171	0.149
	Rocca-S (AD only)	0.478	0.172	0.151
	AEGIS128 (AD only)	0.619	0.401	0.389
	AEGIS128L (AD only)	0.416	0.208	0.195
	Tiaoxin-346 v2 (AD only)	0.328	0.131	0.121
	Jean-Nikolić	0.307	0.126	0.113
	LeMac	0.289	0.082	0.068
	PetitMac	0.521	0.384	0.376
AMD Zen1 (EPYC 7301)	GCM (AD only)	0.932	0.567	0.538
	Rocca (AD only)	0.431	0.147	0.127
	Rocca-S (AD only)	0.438	0.159	0.142
	AEGIS128 (AD only)	0.508	0.325	0.376
	AEGIS128L (AD only)	0.376	0.177	0.181
	Tiaoxin-346 v2 (AD only)	0.358	0.142	0.129
	Jean-Nikolić	0.338	0.125	0.112
	LeMac	0.330	0.088	0.076
	PetitMac	0.670	0.511	0.501
AMD Zen3 (EPYC 7513)	GCM (AD only)	0.794	0.470	0.451
	Rocca (AD only)	0.393	0.139	0.124
	Rocca-S (AD only)	0.417	0.157	0.141
	AEGIS128 (AD only)	0.502	0.339	0.329
	AEGIS128L (AD only)	0.358	0.183	0.173
	Tiaoxin-346 v2 (AD only)	0.311	0.121	0.109
	Jean-Nikolić	0.312	0.111	0.098
	LeMac	0.309	0.085	0.072
	PetitMac	0.655	0.510	0.501

0.125 cycle per byte, while the Tiger Lake and Zen 3 architectures compute at most 2 AES rounds per cycle, corresponding to a limit of 0.0625 cycle per byte. Tiaoxin, Rocca, and

the Jean-Nikolić round function also have rate 2, but they don't allow enough parallelism to reach this bound.

We have also implemented and benchmarked PetitMac on a microcontroller setting. More precisely, our benchmarks were run on the STM32F407VG microcontroller, which is based on the ARM Cortex-M4 processor. For the AES round implementation we used the T-table-based one written in ARMv7-M assembly from [SS16] while we implemented the round function in C code. The code was compiled using `arm-none-eabi-gcc 10.3.1` with the `-O3` optimization flag and the processor was clocked at 24MHz to take advantage of zero wait-states. Processing 16384-byte messages required 299509 clock cycles (witout the initialization and finalization), leading to 18.3 c/B. This performance places PetitMac as a very competitive MAC on microcontrollers, even though it was not directly designed for that platform (AES round is probably not the best starting point).

As expected, PetitMac is not competitive on high-end desktop, because we have to perform two sequential AES rounds per input block, and the latency of the AES instruction is the bottleneck.

8 Conclusion

In this article, we introduced a novel family of extremely fast UHFs, optimized for servers/desktop computers with AES-NI. Our general construction is large enough to offer high granularity to contain interesting security/performance tradeoffs, while ensuring a manageable automated security analysis with MILP. Our strategy to search for good candidates within this family is fully automated and adaptable to the performance profiles of future processors. We showcased the validity of our approach by proposing concrete UHFs and corresponding MACs schemes, largely improving over the state-of-the-art on recent processors. Notably, our proposal LeMac is currently by far the fastest MAC on the high-profile use-case of AES-NI platforms.

Acknowledgments

The authors would like to thanks Alexandre Adomnicai for improving our PetitMac implementation on 32-bit platforms and benchmarking them on the ARM Cortex-M4 microcontroller. Intel/AMD benchmarks were carried out using the Grid'5000 testbed. This work was supported by the bilateral NRF/ANR grant SELECT (NRF-NRFI08-2022-0013/ANR-20-CE48-0017). Thomas Peyrin is supported by the Singapore NRF Investigatorship grant (NRF-NRFI08-2022-0013). Augustin Bariant is supported by the French DGA. Léo Perrin is supported by the European Research Council (ERC, grant agreement no. 101041545 "ReSCALE"). This work was also supported by project Cryptanalyse from PEPR Cybersécurité (22-PECY-0010).

References

[BBG+08] Ryad Benadjila, Olivier Billet, Henri Gilbert, Gilles Macario-Rat, Thomas Peyrin, Matt Robshaw, and Yannick Seurin. SHA-3 Proposal: ECHO. Submission to NIST SHA-3 Cryptographic Hash Algorithm Competition, 2008. Available at https://ehash.iaik.tugraz.at/uploads/9/91/Echo.pdf.

[BD08] Eli Biham and Orr Dunkelman. The shavite-3 hash function. Submission to NIST SHA-3 Cryptographic Hash Algorithm Competition, 2008. Available at https://www.cs.rit.edu/~ark/20090927/Round2Candidates/SHAvite-3.pdf.

[BDPA06] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Radiogatún, a belt-and-mill hash function. Cryptology ePrint Archive, Report 2006/369, 2006. https://eprint.iacr.org/2006/369.

[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, Paris, France, February 21–23, 2005. Springer, Heidelberg, Germany.

[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, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany.

[BLLS22] Jannis Bossert, Eik List, Stefan Lucks, and Sebastian Schmitz. Pholkos - efficient large-state tweakable block ciphers from the AES round function. In Steven D. Galbraith, editor, *CT-RSA 2022*, volume 13161 of *LNCS*, pages 511–536, Virtual Event, March 1–2, 2022. Springer, Heidelberg, Germany.

[CHP+17] Carlos Cid, Tao Huang, Thomas Peyrin, Yu Sasaki, and Ling Song. A security analysis of Deoxys and its internal tweakable block ciphers. *IACR Trans. Symm. Cryptol.*, 2017(3):73–107, 2017.

[CS16] Benoît Cogliati and Yannick Seurin. EWCDM: An efficient, beyond-birthday secure, nonce-misuse resistant MAC. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 121–149, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[CW77] J Lawrence Carter and Mark N Wegman. Universal classes of hash functions. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 106–112. ACM, 1977.

[DC98] Joan Daemen and Craig S. K. Clapp. Fast hashing and stream encryption with PANAMA. In Serge Vaudenay, editor, *FSE'98*, volume 1372 of *LNCS*, pages 60–74, Paris, France, March 23–25, 1998. Springer, Heidelberg, Germany.

[DF04] David Steven Dummit and Richard M. Foote. *Abstract Algebra*. John Wiley & sons, Hoboken, NJ, Third edition, 2004.

[DMN23] Christoph Dobraunig, Bart Mennink, and Samuel Neves. Elimac: Speeding up lightmac by around 20%. *IACR Trans. Symmetric Cryptol.*, 2023(2):69–93, 2023.

[DR02] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES — the Advanced Encryption Standard*. Springer-Verlag, 2002.

[Dwo07] Morris Dworkin. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. Technical report, National Institute of Standards and Technology, November 2007. NIST Special Publication 800-38D.

[FJP13] Pierre-Alain Fouque, Jérémy Jean, and Thomas Peyrin. Structural evaluation of AES and chosen-key distinguisher of 9-round AES-128. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 183–203, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[Fog22] Agner Fog. Lists of instruction latencies, throughputs and micro-operation breakdowns for intel, amd, and via cpus, 2022.

[Gan90] Felix Ruvimovich Gantmacher. *The Theory of Matrices*, volume 1. Chelsea Publishing Company, Second edition, 1990.

[GK08] Shay Gueron and Michael E. Kounavis. Vortex: A new family of one-way hash functions based on aes rounds and carry-less multiplication. In Tzong-Chen Wu, Chin-Laung Lei, Vincent Rijmen, and Der-Tsai Lee, editors, *Information Security*, pages 331–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[GM16] Shay Gueron and Nicky Mouha. Simpira v2: A family of efficient permutations using the AES round function. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 95–125, Hanoi, Vietnam, December 4–8, 2016. Springer, Heidelberg, Germany.

[Gue08] Shay Gueron. Intel Advanced Encryption Standard (AES) New Instructions Set. White Paper Rev 3.01 (09/2012), Intel Corporation, 2008.

[Gur23] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

[IAC+08] Sebastiaan Indesteege, Elena Andreeva, Christophe De Cannière, Orr Dunkelman, Emilia Käsper, Svetla Nikova, Bart Preneel, and Elmar Tischhause. The Lane hash function. Submission to NIST SHA-3 Cryptographic Hash Algorithm Competition, 2008. Available at https://www.cosic.esat.kuleuven.be/lane/index.shtml.

[IIL+23] Takanori Isobe, Ryoma Ito, Fukang Liu, Kazuhiko Minematsu, Motoki Nakahashi, Kosei Sakamoto, and Rentaro Shiba. Areion: Highly-efficient permutations and its applications to hash functions for short input. *IACR TCHES*, 2023(2):115–154, 2023.

[Int24] Intel. Optimizing earlier generations of intel® 64 and ia-32 processor architectures, throughput, and latency, 2024.

[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, Bochum, Germany, March 20–23, 2016. Springer, Heidelberg, Germany.

[JNPS21] Jérémy Jean, Ivica Nikolic, Thomas Peyrin, and Yannick Seurin. The deoxys AEAD family. *Journal of Cryptology*, 34(3):31, July 2021.

[KLMR16] Stefan Kölbl, Martin M. Lauridsen, Florian Mendel, and Christian Rechberger. Haraka - efficient short-input hashing for post-quantum applications. Cryptology ePrint Archive, Report 2016/098, 2016. https://eprint.iacr.org/2016/098.

[KR21] Ted Krovetz and Phillip Rogaway. The design and evolution of OCB. *Journal of Cryptology*, 34(4):36, October 2021.

[KS07] Liam Keliher and Jiayuan Sui. Exact maximum expected differential and linear probability for two-round advanced encryption standard. *IET Inf. Secur.*, 1(2):53–57, 2007.

[MN17] Bart Mennink and Samuel Neves. Encrypted Davies-Meyer and its dual: Towards optimal security using mirror theory. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 556–583, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[MT06] Kazuhiko Minematsu and Yukiyasu Tsunoo. Provably secure MACs from differentially-uniform permutations and AES-based implementations. In Matthew J. B. Robshaw, editor, *FSE 2006*, volume 4047 of *LNCS*, pages 226–241, Graz, Austria, March 15–17, 2006. Springer, Heidelberg, Germany.

[MV04] David A. McGrew and John Viega. The Galois/Counter Mode of Operation (GCM). Submission to NIST Modes of Operation Process, January 2004. Available at https://csrc.nist.rip/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf.

[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.

[NBK08] Ivica Nikolić, Alex Biryukov, and Dmitry Khovratovich. Hash family LUX - Algorithm Specifications and Supporting Documentation. Submission to NIST SHA-3 Cryptographic Hash Algorithm Competition, 2008. Available at https://ehash.iaik.tugraz.at/uploads/f/f3/LUX.pdf.

[NFI24] Yuto Nakano, Kazuhide Fukushima, and Takanori Isobe. Encryption algorithm Rocca-S. Internet-Draft draft-nakano-rocca-s-05, Internet Engineering Task Force, January 2024. Work in Progress.

[Nik14] Ivica Nikolić. Tiaoxin–346. Submission to CAESAR Competition, 2014. Available at https://competitions.cr.yp.to/round3/tiaoxinv21.pdf.

[Nik17a] Ivica Nikolic. How to use metaheuristics for design of symmetric-key primitives. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 369–391, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany.

[Nik17b] Ivica Nikolić. How to use metaheuristics for design of symmetric-key primitives. Cryptology ePrint Archive, Report 2017/851, 2017. https://eprint.iacr.org/2017/851.

[RBB03] Phillip Rogaway, Mihir Bellare, and John Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3):365–403, aug 2003.

[RVPV02] Vincent Rijmen, Bart Van Rompay, Bart Preneel, and Joos Vandewalle. Producing collisions for PANAMA. In Mitsuru Matsui, editor, *FSE 2001*, volume 2355 of *LNCS*, pages 37–51, Yokohama, Japan, April 2–4, 2002. Springer, Heidelberg, Germany.

[Sho96] Victor Shoup. On fast and provably secure message authentication based on universal hashing. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 313–328, Santa Barbara, CA, USA, August 18–22, 1996. Springer, Heidelberg, Germany.

[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.

[SLN+22] Kosei Sakamoto, Fukang Liu, Yuto Nakano, Shinsaku Kiyomoto, and Takanori Isobe. Rocca: An efficient AES-based encryption scheme for beyond 5G (full version). Cryptology ePrint Archive, Report 2022/116, 2022. https://eprint.iacr.org/2022/116.

[SS16] Peter Schwabe and Ko Stoffelen. All the AES you need on cortex-M3 and M4. Cryptology ePrint Archive, Report 2016/714, 2016. https://eprint.iacr.org/2016/714.

[Sti92] Douglas R. Stinson. Universal hashing and authentication codes. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 74–85, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Heidelberg, Germany.

[TSI23] Nobuyuki Takeuchi, Kosei Sakamoto, and Takanori Isobe. On optimality of the round function of rocca. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 106(1):45–53, 2023.

[WP13] Hongjun Wu and Bart Preneel. AEGIS: A fast authenticated encryption algorithm. Cryptology ePrint Archive, Report 2013/695, 2013. https://eprint.iacr.org/2013/695.

[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, Burnaby, BC, Canada, August 14–16, 2014. Springer, Heidelberg, Germany.

A On the speed of Rocca in decryption mode

In the decryption mode of Rocca, we found the following dependency chain from $S[4]$ to $S^{new}[4]$ (with the notation of [SLN+21, Figure 1]), composed of an AESENC and 3 XOR instructions:

$$S^{new}[4] = \mathsf{XOR}(\mathsf{XOR}(C_i^1, \mathsf{AESENC}(\mathsf{XOR}(S[0], S[4]), S[2])), S[3]) \tag{9}$$

We notice that this dependancy chain only appears in the decryption mode, since the value X_1 of [SLN+21, Figure 1] needs to be computed from $S[4]$. On Ice Lake and Tiger Lake architectures, this dependency chain first appears to have a latency of 6 cycles, which implies that the round function can not be faster than 6 cycles per 2×128 bits of message, or equivalently 0.19 cycles per byte.

Bypass delay We measured the performance of the round function of Rocca in decryption mode on Tiger Lake (i5-1135G7) and measure around 0.34 cycles per byte. We believe that this corresponds to a latency of around 10 cycles. As far as we can tell, there is an additional delay (a *bypass delay*) when the output of an AES instruction is used as input to a non-AES instruction. In particular, the latency of the dependency chain of Equation 9 increases by 4 cycles (to a total of 10 cycles of latency): the two XOR instructions on the left of Equation 9 take in input respectively $S[3]$ and $\mathsf{AESENC}(\mathsf{XOR}(S[0], S[4]), S[2])$ which are both output of an AESENC instruction. The latency can be reduced to 8 cycles with a fake XOR instruction added at the beginning of the round : $S[3] \leftarrow \mathsf{XOR}(S[3], 0x00)$, so that at the beginning of each round, $S[3]$ is not the direct result of an AESENC encryption. This was tested experimentally and increases the speed to around 0.25 cycles per byte on Tiger Lake (i5-1135G7). It seems that these 8 cycles of latency can not be reduced, and we therefore believe that Rocca in the decryption mode can not run faster than $8/(2 \times 16) = 0.25$ cycles per byte on Tiger Lake processors.

B Full Proof for Reducing the Search Space

Proof of Proposition 1. Let us denote for any $t \geqslant 0$, R_P^t, V_P^t the respective memory registers and round-message at clock t produced by T_P. By adapting Equation (3) to T_P, we obtain:

$$\forall t \geqslant 0 \quad R_P^{t+1} = PT_{00}P^{-1}R_P^t + PT_{01}M^t \qquad V_P^t = T_{10}P^{-1}R_P^t + T_{11}M^t. \tag{10}$$

By design, $R^0 = 0$ and $R_P^0 = 0$ because the memory is initialized as such. In particular $R_P^0 = PR^0$. Let $t \geqslant 0$ and let suppose that $R_P^t = PR^t$. Then by injecting $R_P^t = PR^t$ into Equation (10) and simplifying we get,

$$\begin{aligned} R_P^{t+1} &= PT_{00}P^{-1}R_P^t + PT_{01}M^t \\ &= PT_{00}P^{-1}PR^t + PT_{01}M^t \\ &= PT_{00}R^t + PT_{01}M^t \\ &= P(T_{00}R^t + T_{01}M^t) = PR^{t+1}. \end{aligned}$$

This first proves by induction that $R_P^t = PR^t$ for any $t \geqslant 0$.

Let $t \geqslant 0$. According to Equation (10), $V_P^t = T_{10}P^{-1}R_P^t + T_{11}M^t$. Replacing R_P^t by PR^t, we obtain:

$$V_P^t = T_{10}P^{-1}PR^t + T_{11}M^t = T_{10}R^t + T_{11}M^t = V^t;$$

which proves the announced equality for any $t \geqslant 0$. □

Proof of Corollary 1. If $d = r$ then T generates $\mathcal{V}$ and has d memory-registers. Let us now suppose that $d < r$. In that case, we can find $P \in \mathrm{GL}_r(\mathbb{F}_2)$ such that the first $r - d$ rows of $(PT_{00}|PT_{01}) = P(T_{00}|T_{01})$ are all-0. $(PT_{00}P^{-1}|PT_{01})$ naturally shares the same property, and according to Proposition 1, T_P produces the same round-message sequence. But the $r - d$ first empty rows in T_P indicates that the first $r - d$ memory registers will be zero at all time $t \geqslant 0$, and therefore will never impact the output sequence. T_P can thus be adapted by removing the $r - d$ null rows in the upper half, and removing the corresponding $r - d$ columns in the left-hand half. The obtained matrix T' generates the same sequence with d memory registers. □

Proof of Proposition 4. By hypothesis, $\mathsf{rank}(T_{11}) = m$, so at each round, the information of the m independent new message blocks is fully contained in m of the round-value blocks. In other words, there exists m indices $I = \{i_0, \cdots, i_{m-1}\}$ such that for any t, $V_I^t = \begin{pmatrix} F & C \end{pmatrix} \times \mathcal{M}^{[\![0,t]\!]}$, where $C \in \mathrm{GL}_m(\mathbb{F}_2)$ (and $F \in \mathcal{M}_{m\times(t-1)m}(\mathbb{F}_2)$).

Up to a wire permutation, let us assume that $I = [\![0, m-1]\!]$. In that case, $(T_{10}|T_{11})$ can be decomposed, such that C appears in it:

$$\begin{pmatrix} T_{10} & T_{11} \end{pmatrix} = \begin{pmatrix} \overset{r}{\leftrightarrow} & \overset{m}{\leftrightarrow} \\ B & C \\ D & E \end{pmatrix} \begin{matrix} \updownarrow m \\ \updownarrow s-m \end{matrix}. \tag{11}$$

Now, let $\ell \in \mathbb{N} \setminus \{0\}$ and let us consider the following change of variables:

$$\forall t \in [\![0, \ell-1]\!], \quad \widetilde{M}^t := BR^t + CM^t \iff C^{-1}(\widetilde{M}^t - BR^t) = M^t.$$

Because R^t is a linear combination of $M_i^{t'}$ where $t' < t$, this change of variables corresponds to a lower triangular block matrix P^t (whose diagonal is only made of C blocks).

Decomposing V^t as $V^t = (V^t_{[\![0,m-1]\!]} \mid V^t_{[\![m,s-1]\!]})$, we can rewrite the linear relations in Equation (3) using the decomposition of $(T_{10}|T_{11})$ given in Equation (11):

$$R^0 = 0, \quad \forall t \geqslant 0,\ R^{t+1} = T_{00}R^t + T_{01}M^t,$$
$$\forall t \geqslant 0,\ V^t_{[\![0,m-1]\!]} = BR^t + CM^t, \quad V^t_{[\![m,s-1]\!]} = DR^t + EM^t.$$

By substituting M^t in the previous equations we obtain:

$$R^0 = 0, \quad \forall t \geqslant 0,\ R^{t+1} = T_{00}R^t + T_{01}C^{-1}(\widetilde{M}^t - BR^t),$$
$$V^t_{[\![0,m-1]\!]} = BR^t + CC^{-1}(\widetilde{M}^t - BR^t), \quad V^t_{[\![m,s-1]\!]} = DR^t + EC^{-1}(\widetilde{M}^t - BR^t);$$

which, once simplified and reorganized, become:

$$R^0 = 0, \quad \forall t \geqslant 0,\ R^{t+1} = (T_{00} - T_{01}C^{-1}B)R^t + T_{01}C^{-1}\widetilde{M}^t,$$
$$V^t_{[\![0,m-1]\!]} = \widetilde{M}^t, \quad V^t_{[\![m,s-1]\!]} = (D - EC^{-1}B)R^t + EC^{-1}\widetilde{M}^t.$$

Thus, the sequence $\mathcal{V}$, is linearly-equivalent to the sequence generated by the transition matrix $\widetilde{T}$ defined as:

$$\widetilde{T} := \left(\begin{array}{c|c} T_{00} - T_{01}C^{-1}B & T_{01}C^{-1} \\ \hline 0 & \mathrm{Id}_m \\ D - EC^{-1}B & EC^{-1} \end{array}\right). \tag{12}$$

$\widetilde{T}$ has the announced form. □

C Candidates

C.1 7 AES per round (rate=1.75)

$$L = \begin{bmatrix} 0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0 \\ 0&0&1&0&0&0&0&0&0&1 \\ 0&1&0&0&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0 \end{bmatrix} \quad T = \left[\begin{array}{ccccc|cccc} 0&0&0&0&0&0&0&1&0 \\ 1&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&1 \\ 0&0&1&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \\ \hline 0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0&0 \\ 0&1&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \end{array}\right] \quad A = (1100111110) \quad r = 5 \quad s = 10 \quad m = 4 \ .$$

C.2 6 AES per round

$$L = \begin{bmatrix} 0&1&0&0&0&0&0 \\ 0&1&0&1&0&0&0 \\ 1&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1 \\ 0&0&1&0&0&0&0 \end{bmatrix} \quad T = \left[\begin{array}{cccc|ccc} 0&0&0&0&0&1&1 \\ 1&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&1&0&0&0&0 \\ \hline 0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1 \\ 0&0&0&1&0&0&0 \\ 0&0&0&0&1&0&1 \\ 0&1&0&0&0&0&0 \end{array}\right] \quad A = (1011111) \quad r = 4 \quad s = 7 \quad m = 3 \ .$$

C.3 4 AES per round

$$L = \begin{bmatrix} 1&1&0&0&0&0 \\ 0&0&0&0&1&0 \\ 0&0&0&0&0&1 \\ 1&0&0&0&0&0 \\ 0&0&1&0&0&0 \\ 0&0&0&1&0&0 \end{bmatrix} \quad T = \left[\begin{array}{cccc|cc} 0&0&0&1&1&0 \\ 1&0&0&0&0&0 \\ 0&1&0&0&0&0 \\ 0&0&1&0&0&0 \\ \hline 0&0&0&0&1&0 \\ 0&0&0&0&0&1 \\ 0&0&0&0&0&1 \\ 0&0&0&0&1&0 \\ 1&0&0&0&0&0 \\ 0&0&0&0&0&0 \end{array}\right] \quad A = (011110) \quad r = 4 \quad s = 6 \quad m = 2 \ .$$

C.4 2 AES per round

$$L = \begin{bmatrix} 0&0&1&0 \\ 1&0&0&0 \\ 0&0&1&1 \\ 0&1&0&0 \end{bmatrix} \quad T = \left[\begin{array}{ccc|c} 0&0&1&1 \\ 1&0&0&0 \\ 0&1&0&0 \\ \hline 0&0&0&1 \\ 0&0&0&1 \\ 0&0&0&1 \\ 0&0&0&1 \end{array}\right] \quad A = (1100) \quad r = 3 \quad s = 4 \quad m = 1 \ .$$

D Algorithms

D.1 LeMac

A complete description of LeMac is provided in Algorithm 1, where A is a key-less AES round. All the subkeys are derived by encrypting a counter with the master key. The state of the UHF is initialized with such subkeys. During the UHF finalization, each branch of the inner state goes through 10 independent AES rounds with subkeys that were derived as encrypted counters. We derive only 18 subkeys, and use those in a rolling fashion in each different branch. The idea is to save space by not having to store $10 \times s = 90$ different 128-bit subkeys for this final step. Figure 4 shows five rounds of the UHF used in LeMac.

D.2 PetitMac

As with LeMac, we give a complete description of PetitMac in Algorithm 2. The notations are the same as above. All the subkeys are derived by encrypting a counter with the master key. The state of the UHF is initialized with one such subkey. During the UHF finalization, we encrypt the state and the memory registers using 10-round AES, with

Algorithm 1 LeMac, with 128-bit master key K, 128-bit nonce N, and message M.

▷ Key derivation
$\mathsf{Kinit} \leftarrow (\mathsf{AES}_K(0), ...\mathsf{AES}_K(8))$
$\mathsf{Kfinal} \leftarrow (\mathsf{AES}_K(9), ...\mathsf{AES}_K(26))$
$k_2 \leftarrow \mathsf{AES}_K(27)$
$k_3 \leftarrow \mathsf{AES}_K(28)$
▷ UHF
$X^0 \leftarrow \mathsf{Kinit}$ ▷ Start of initialization
$R^0 \leftarrow (0, 0, 0)$
▷ Padding
$\ell \leftarrow \lceil (\text{bitlen}(M) + 1)/512 \rceil$
$M_0, ..., M_{4\ell-1} \leftarrow M \| 10^*$
$M_{4\ell}, ..., M_{4\ell+11} \leftarrow 0$
for all $0 \leqslant i < \ell + 3$ **do** ▷ Start of absorption (see Figure 2)
  $X_0^{i+1} \leftarrow X_0^i + X_8^i$
  for all $1 \leqslant j < 9$ **do**
    $X_j^{i+1} \leftarrow A(X_{j-1}^i)$
  end for
  $X_0^{i+1} \leftarrow X_0^{i+1} + M_{4i+2}$
  $X_1^{i+1} \leftarrow X_1^{i+1} + M_{4i+3}$
  $X_2^{i+1} \leftarrow X_2^{i+1} + M_{4i+3}$
  $X_3^{i+1} \leftarrow X_3^{i+1} + R_1^i + R_2^i$
  $X_4^{i+1} \leftarrow X_4^{i+1} + M_{4i}$
  $X_5^{i+1} \leftarrow X_5^{i+1} + M_{4i}$
  $X_6^{i+1} \leftarrow X_6^{i+1} + M_{4i+1}$
  $X_7^{i+1} \leftarrow X_7^{i+1} + M_{4i+1}$
  $X_8^{i+1} \leftarrow X_8^{i+1} + M_{4i+3}$
  $R_0^{i+1} \leftarrow M_{4i+2}$
  $R_1^{i+1} \leftarrow R_0^i + M_{4i+1}$
  $R_2^{i+1} \leftarrow R_1^i$
end for
for all $0 \leqslant j < 9$ **do** ▷ Start of finalization
  for all $0 \leqslant i < 10$ **do**
    $X_j^{\ell+3+i+1} \leftarrow A\left(X_j^{\ell+3+i} + \mathsf{Kfinal}_{i+j}\right)$
  end for
end for
$h \leftarrow \sum_{j=0}^{8} X_j^{\ell+13}$
▷ EWCDM
return $\mathsf{AES}_{k_3}\left(h + \mathsf{AES}_{k_2}(N) + N\right)$

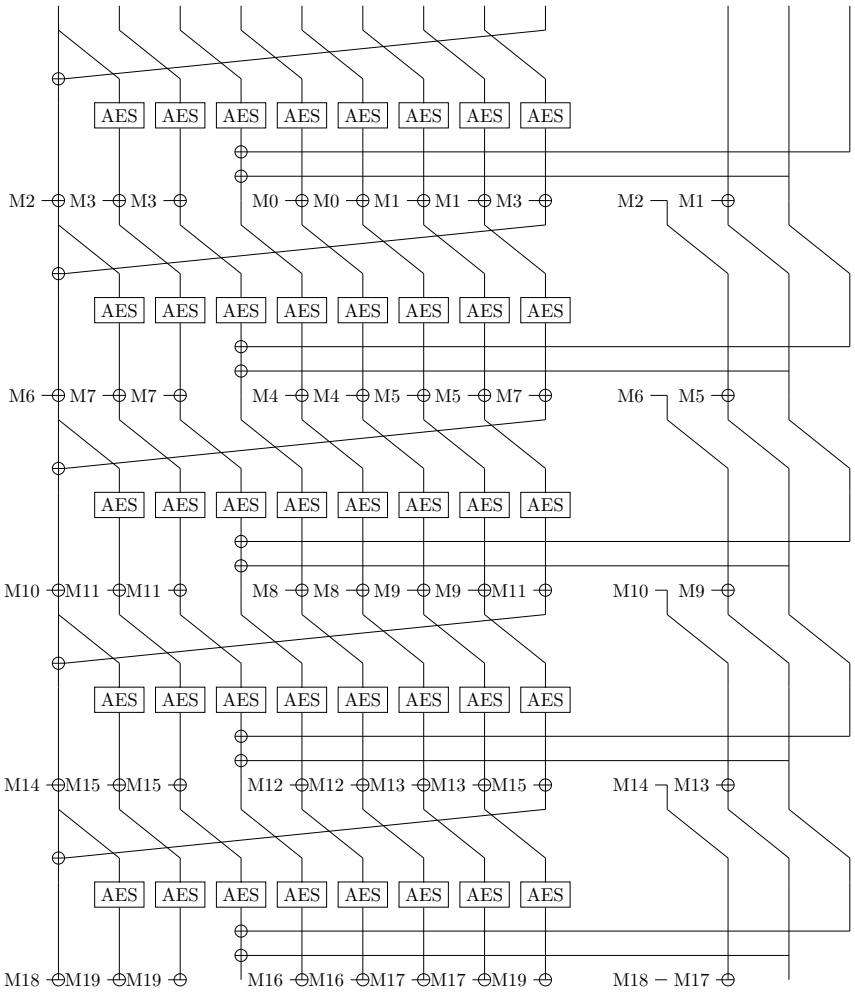

Figure 4: Five rounds of the UHF used in LeMac.

subkeys generated in a rolling fashion. We XOR the outputs together, and combine the result with the nonce using the EWCDM construction.

Algorithm 2 PetitMac, with 128-bit master key K, 128-bit nonce N, and message M.

▷ Key derivation
$\mathsf{Kinit} \leftarrow \mathsf{AES}_K(0)$
$\mathsf{Kfinal} \leftarrow (\mathsf{AES}_K(1), ...\mathsf{AES}_K(15))$
$k_2 \leftarrow \mathsf{AES}_K(16)$
$k_3 \leftarrow \mathsf{AES}_K(17)$
▷ UHF
$X^0 \leftarrow \mathsf{Kinit}$ ▷ Start of initialization
$R^0 \leftarrow (0,0,0,0,0)$
▷ Padding
$\ell \leftarrow \lceil (\text{bitlen}(M)+1)/128 \rceil$
$M_0, ..., M_{\ell-1} \leftarrow M \| 10^*$
for all $0 \leqslant i < \ell$ **do** ▷ Start of absorption (see Figure 3)
  $t \leftarrow A(X^i) + M_i + R_4^i$
  $R_0^{i+1} \leftarrow M_i + R_3^i$
  $R_1^{i+1} \leftarrow R_4^i + R_0^{i+1}$
  $R_2^{i+1} \leftarrow R_4^i + R_0^i$
  $R_3^{i+1} \leftarrow R_1^i$
  $R_4^{i+1} \leftarrow R_2^i$
  $X^{i+1} \leftarrow A(t) + R_0^{i+1}$
end for ▷ Finalization
for all $0 \leqslant i < 10$ **do** ▷ Start of finalization
  $X^{\ell+i+1} \leftarrow A\left(X^{\ell+i} + \mathsf{Kfinal}_i\right)$
end for
for all $0 \leqslant j < 5$ **do**
  for all $0 \leqslant i < 10$ **do**
    $R_j^{\ell+i+1} \leftarrow A\left(R_j^{\ell+i} + \mathsf{Kfinal}_{i+j+1}\right)$
  end for
end for
$h \leftarrow X^{\ell+10} + \sum_{j=0}^{4} R_j^{\ell+10}$
▷ EWCDM
return $\mathsf{AES}_{k_3}\left(h + \mathsf{AES}_{k_2}(N) + N\right)$