New Results on Romulus

Tetsu Iwata[1], Mustafa Khairallah[2], Kazuhiko Minematsu[4], and Thomas Peyrin[2,3]

[1] Nagoya University, Nagoya, Japan
tetsu.iwata@nagoya-u.jp
[2] Nanyang Technological University, Singapore, Singapore
[3] Temasek Laboratories@NTU, Singapore, Singapore
mustafam001@e.ntu.edu.sg,thomas.peyrin@ntu.edu.sg
[4] NEC, Kawasaki, Japan
k-minematsu@nec.com

Keywords: Romulus· Hardware implementation· Release of unverified plaintext· Leakage Resilience· Hash function.

Abstract. Romulus is a family of lightweight authenticated encryption with associated data (AEAD) algorithms based on the extensively-studied ultra-lightweight tweakable block cipher (TBC) Skinny. It was designed to be small and efficient for constrained implementations while achieving 128-bit provable security based on standard security assumptions. In this article, we present new contents, analysis, proofs, benchmarks relative to Romulus submission.

- we decided to simplify our submission by keeping only the versions based on Skinny-128/384 as we believe they are the most interesting (the nonce-respecting mode is now named Romulus-N, the nonce-misuse resistant mode is now named Romulus-M). Moreover, as publicly announced by the Skinny team, due to the huge security margin offered by Skinny-128/384, the number of rounds has been decreased from 56 to 40 (now named Skinny-128/384+), which still maintains more than 30% security margin even for near-brute force complexity distinguishers. This will directly offer a 40% boost on all hardware/software performance figures currently displayed for Romulus.
- we have added a simple and efficient rate-1 TBC-based hash function/XOF capability Romulus-H, by directly using Naito's provably-secure MDPH construction: the combination of the well-known Hirose DBL scheme and the Merkle-Damgård with Permutation (MDP) domain extender. We also designed leakage-resilient AEAD modes (Romulus-LR and Romulus-LR-TEDT for two levels of leakage resilience), both coming with a proof in the leakage model. We emphasize that both these modes are using the same internal primitive Skinny-128/384+ and are themselves very similar (for example Romulus-LR simply re-injects the message in the tweak inputs compared to Romulus-N).
- we have added a proof for the INT-RUP security notion as well as the plaintext-awareness PA1 security notion for Romulus-M.
- we have provided threshold implementations of Romulus, which are very competitive according to publicly reported threshold implementations of other candidates. As shown by Naito *et al.* at Eurocrypt 2020, TBCs have a great small-size advantage over other designs for such implementations.
- we studied various performance trade-offs in hardware implementations, which show that the Romulus design offers excellent security-performance-area trade-offs.

1 Introduction

Romulus is a family of lightweight Authenticated Encryption with Associated Data (AEAD) algorithms based on Tweakable Block Ciphers (TBC) designed by Iwata et al. [IKMP19, IKMP20]. It has been selected as a second round candidate for the NIST lightweight cryptography standardization process. The Romulus family of algorithms was designed to be small and efficient for hardware implementation while achieving 128-bit provable security based on standard security assumptions. It enjoys several attractive features:

No compromise on security:

– The security of Romulus is reduced to the security of the underlying tweakable block cipher, Skinny [BJK+16], based on the standard TPRP assumption (unlike most round 2 candidates of the competition that rely on non-standard ideal-cipher or random-permutation model).

– Romulus achieves provable beyond-birthday-bound (BBB) security (unlike all block-cipher based round 2 candidates of the competition).

– The Romulus family includes nonce-misuse resistant algorithms (unlike most round 2 candidates of the competition) with graceful security degradation when the number of nonce repetitions is small, with birthday security in the unlikely event where the nonce is completely fixed.

– Skinny is a conservative and well-understood SPN design. Since publication, it has been extensively analyzed by both its designers and several third parties (more than 30 third-party cryptanalysis articles, probably the most analysed internal primitive of the competition thus far except AES or KECCAK). The number of attacked rounds remained quite stable since the original Skinny publication. All this raises the confidence level in the security of Skinny. It has been incorporated in the French COVID-19 tracing application under French governmental security agency advice and is currently being considered for standardization at ISO/IEC.

– No attack nor distinguisher exist on the internal primitive Romulus, unlike most sponge-based candidates of the competition (which sometimes even exhibit practical distinguishers by design). The security margin remains very large (about 30% for Skinny-128/384+)

A very efficient all-round lightweight design:

– Romulus is an all-round candidate, which presents excellent hardware performances (both in terms of area and energy/power consumption) and excellent software performances where lightweight cryptography makes the most sense (4-bit, 8-bit processors), as shown in benchmarks.

– Romulus has particularly competitive performance for small messages (crucial for many lightweight applications), unlike many competition candidates that require a long initialisation. For example, 16 bytes of associated data and 16 bytes of message require only two TBC calls for Romulus-N and three TBC calls for Romulus-M.

– As they are all based on the same internal primitive Skinny-128/384+ and as they are very similar to each other, in hardware the various Romulus modes can easily be merged together in the same circuit, allowing flexible implementations with different levels of security.

– As shown by Naito *et al.* at Eurocrypt 2020 [NSS20a], TBCs have a great small-size advantage over other designs for such implementations. This is confirmed by the very competitive threshold implementations of Romulus we report here.

– Generally, Romulus is a very simple algorithm that requires very few resources beyond the underlying TBC. This is confirmed by both our theoretical and practical efficiency comparisons in [IKMP20] and the implementation results shown in this work.

In this article, we give new results on Romulus. Firstly, we chose to simplify the NIST submission by keeping only the main variants based on Skinny-128/384 (renamed Romulus-N and Romulus-M for the occasion). Moreover, as the security margin of Skinny-128/384 is really large, we chose to use instead the round-reduced version Skinny-128/384+ proposed by the Skinny team [Rt20], which still ensures 30% security margin as of today. This allows to directly improve all currently displayed benchmarks by 40% in terms of throughput and latency, while keeping area and memory stable. See Section 2.

Furthermore, we introduce in Section 3 a hash function capability for Romulus, by simply plugging Skinny-128/384+ in Naito's provably-secure MDPH construction [Nai19]. This TBC-based hash function is simply the combination of the well-known Hirose DBL scheme and the Merkle-Damgård with Permutation (MDP) domain extender. It is rate-1 when used with Skinny-128/384+, provides 121-bit collision resistance from the indifferentiability result and can be used as a secure eXtendable Output Function (XOF).

We propose in Section 4 two efficient leakage-resilient AEAD modes (Romulus-LR and Romulus-LR-TEDT for two levels of leakage resilience), both coming with security proofs in leakage model. We emphasize that both these modes are using the same internal primitive Skinny-128/384+ and are very similar to Romulus-N.

In Section 5, we provide a new security analysis for Romulus-M, showing that it can target more misuse scenarios rather than just nonce-misuse, namely achieving INT-RUP security [ABL+14a] naturally. We also cover the privacy notion in the RUP setting called plaintext awareness (PA) [ABL+14a]. Of independent interest, we point out an error in the INT-RUP analysis of SIV shown in the original RUP paper [ABL+14a, ABL+14b].

Finally, in Section 6 we exhibit new optimized implementations for the main variants of Romulus, showing that the design offers excellent security-performance-area trade-offs. In addition, we present very efficient threshold implementations. As already observed in [NSS20a], TBCs have a great advantage with regards to threshold implementations compared to other designs (block cipher-based or sponge-based), since they fundamentally require a smaller state size to protect.

2 Simpler and Faster Variants

The Skinny TBC [BJK+16] went through a lot of third-party analysis efforts over the past 4 years, with more than 30 cryptanalysis papers (and three cryptanalysis competitions conducted), which makes it the most analysed primitive of the competition, except AES or KECCAK. The number of attacked rounds stayed quite stable since the Skinny original publication. This confidence led to the incorporation of Skinny in the French COVID-19 tracing application under French governmental security agency advice. Moreover, Skinny is currently being considered for standardization at ISO/IEC (currently working draft).

At time of writing, the best known attacks against Skinny-128/384 cover 28 rounds [ZDM+19] (out of 56 rounds), which means that the security margin is of 50% and actually much more if one considers only single-key attacks and/or attacks with a complexity lower than 2^{128}. Indeed, all these attacks have very high complexity, much more than 2^{200} in computational complexity and sometimes up to almost 2^{384}, and only work in the related-tweakey model where differences need to also be inserted in the tweak and/or key input. In the single-key model, the best known attacks against Skinny-128/384 covers only 22 rounds [TAY17, SSD+18, CSSH19], again all these attacks having a very high computational complexity.

Compared to other block ciphers, where the security margin is usually at very best around 33%, this security margin is maybe too large. Even more so if we compare with permutations used in sponge functions proposals, where non-random behaviour can be usually exhibited for the full-round internal primitive for a complexity lower than the targeted security parameter of the whole scheme (actually often with practical complexity). For this reason, the Skinny team decided to propose a new variant of Skinny-128/384 (named Skinny-128/384+) by reducing its number of rounds from 56 to 40, to give a security margin of around 30% (in the worst-case related-tweakey scenario, without even excluding attacks with complexity much higher than 2^{128}), which still provides a very large security margin [Rt20].

In order to simplify our NIST submission, we decided to only keep the Romulus versions based on Skinny-128/384+ (and thus now Skinny-128/384+), which were our original primary versions. Indeed, we believe these versions are the most interesting ones (for the same performance, they offer more flexibility) and in addition our submission gains in consistency as now **all** our modes are based on Skinny-128/384+ only. To summarize, Romulus will now consists in a nonce-respecting AEAD mode Romulus-N, a nonce-misuse resistant AEAD mode Romulus-M, two leakage resilient AEAD modes Romulus-LR, Romulus-LR-TEDT, and one hash function/XOF Romulus-H.

We emphasize that Romulus-N and Romulus-M are exactly the same as previous round Romulus-N1 and Romulus-M1, with simply Skinny-128/384+ used instead of Skinny-128/384. The security claims remain of course the same, while there will provide a 40% direct performance throughput/latency improvement over currently reported benchmarks (for the same area/memory). In Table 1, we summarize the specification of these new members and compare them with old variants.

Table 1: New and old variants of Romulus.

New members of Romulus for round 3.

Member	Mode	Primitive	Comment
Romulus-N	Romulus-N1 [IKMP19, IKMP20]	Skinny-128/384+	BBB nonce-respecting AEAD
Romulus-M	Romulus-M1 [IKMP19, IKMP20]		BBB nonce-misuse resistant AEAD
Romulus-H	MDPH [Nai19]		Hash function / XOF
Romulus-LR	AET-LR [GKP20]		leakage resilient AEAD (CIML2 + CCAml1)
Romulus-LR-TEDT	TEDT [BGP+19]		leakage resilient AEAD (CIML2 + CCAmL2)

Previous members of Romulus for round 1 and 2.

Member	Mode	Primitive	Comment
Romulus-N1	Romulus-N1 [IKMP19, IKMP20]	Skinny-128/384	BBB nonce-respecting AEAD
Romulus-N2		Skinny-128/384	BBB nonce-respecting AEAD
Romulus-N3		Skinny-128/256	BBB nonce-respecting AEAD
Romulus-M1	Romulus-M1 [IKMP19, IKMP20]	Skinny-128/384	BBB nonce-misuse resistant AEAD
Romulus-M2		Skinny-128/384	BBB nonce-misuse resistant AEAD
Romulus-M3		Skinny-128/256	BBB nonce-misuse resistant AEAD

3 Romulus-H: Hashing with Romulus

Since hashing capability was not originally added in the Romulus submission and since this can be achieved quite naturally, we have formalised Romulus-H: a hash function based on Skinny-128/384+. It is simply Skinny-128/384+ placed into Naito's MDPH construction [Nai19], which consists of Hirose's Double-Block-Length (DBL) compression function [Hir06] plugged into the Merkle-Damgård with Permutation (MDP) domain extender [HPY12]. The full Romulus-H is depicted in Figure 1, while the formal specification is given in Figure 2.

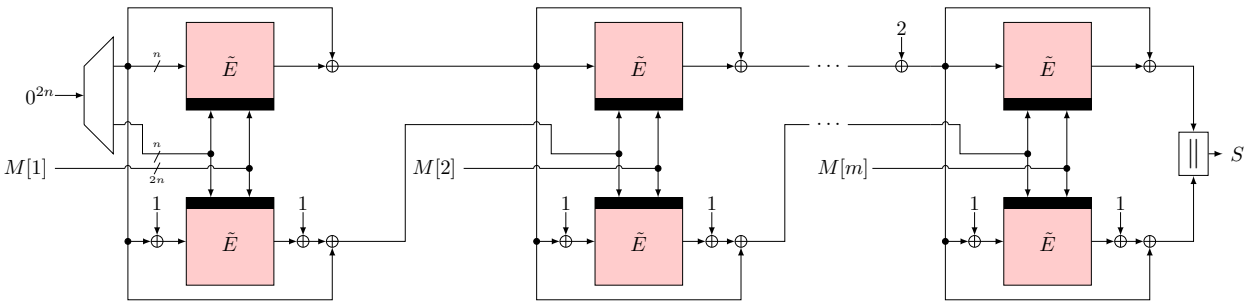


Fig. 1: Block diagram of Romulus-H hash function.

This construction is proven secure: when the output is $2n$ bits, MDPH is indifferentiable [MRH04] from a (variable-input-length) random oracle up to about $(n - \log n)$ queries [Nai19] assuming the ideal (tweakable) block cipher. We set $n = 128$ and immediately have 121-bit indifferentiability. The standard reduction [AMP10] tells that Romulus-H is proved to have 121-bit `atk`-security for any of `atk` $\in$ {`collision`, `preimage`, `2nd-preimage`}. Note that indifferentiability is a very useful and versatile security notion for hash functions that some classical collision-resistant constructions fail to meet [CDMP05]. Romulus-H can be easily turns into an eXtendable Output Function (XOF) that has an arbitrarily long output. This is

because MDPH is indifferentiable from a (monolithic) random oracle, thus any black-box transformation that turns a RO into a XOF will also work for MDPH. One simple example is to use $H(M\|[0]), H(M\|[1]), \ldots$, where H is the base hash function and $[i]$ denotes an encoding of integer i. In fact this is just a variant of standard MGF1 (Mask generation function). Additional computation cost of this transformation is small (one compression function call per output block) thanks to the iterative nature of MDPH.

Algorithm Romulus-H(M)

1. $S \leftarrow 0^{2n}$
2. $M \leftarrow \mathsf{pad}(M)$
3. $M[1], \ldots, M[m] \xleftarrow{2n} M$
4. **for** $i = 1$ **to** $m - 1$ **do**
5. $\quad S \leftarrow \mathsf{CF}(S, M[i])$
6. $S \leftarrow S \oplus (2 \,\|\, 0^n)$
7. $S \leftarrow \mathsf{CF}(S, M[m])$
8. **return** S

Algorithm CF(X, Y)

1. $L, R \xleftarrow{n} X$
2. $S \leftarrow \widetilde{E}^{Y,R}(L) \oplus L$
3. $V \leftarrow \widetilde{E}^{Y,R}(L \oplus 1) \oplus L \oplus 1$
4. $Z \leftarrow S \,\|\, V$
5. **return** Z

Fig. 2: Definition of Romulus-H. CF is Hirose's DBL compression function. The whole hashing function follows Naito's MDPH. pad : $\{0,1\}^* \rightarrow (\{0,1\}^{2n})^+$ is an injective padding function.

In terms of performances, Hirose's scheme requires two TBC calls, but since we are using Skinny-128/384+, 256-bit message blocks can be handled at each iteration, which makes Romulus-H an efficient rate-1 construction overall. We note that this is three times more efficient than the Skinny-Hash construction [BJK+19] (where Skinny-128/384 is used inside a standard sponge-based mode), for the same area cost. Moreover, the fact that each pair of TBC calls have the same tweakey input, combined with the lightweight tweakey schedule of Skinny, is helpful to achieve efficient implementations of this construction as the tweakey can be recovered and stored only once if only one Skinny-128/384+ hardware core is available. If two cores are available, they can share the same round keys. Moreover, we note that the hash can be naturally adapted to extremely constrained area environments by reducing the message input at every iteration (this is possible because Romulus-H places the message input in the tweak input of the TBC, and because Skinny-128/384+ tweakey schedule can be totally replaced by constants if some words are set to 0).

4 Romulus-LR and Romulus-LR-TEDT: Leakage-Resilient Modes for Romulus

Even though we provide efficient threshold implementations of Romulus, we studied how leakage resilience capability could be added to our candidate. It turns out that this can be achieved with a very simple modification of the Romulus mode. More precisely, we will propose two modes for leakage resilience: Romulus-LR and Romulus-LR-TEDT.

Romulus-LR is the first mode, which simply consists in (a) adding a key-derivation function (KDF) at the beginning of Romulus-N, to generate a temporary key K' that will be used in the subsequent TBC calls (b) re-injecting the message blocks inside the tweak input of each TBC call. It is then expected that the KDF and tag generating function (TGF), both using the master key K, should be properly protected with side-channels attacks countermeasures (such as masking). However, the long chain that depends on the message or associated data blocks can be left unprotected (or with much cheaper protection), which leads to a very efficient design (close to the original Romulus-N or Romulus-M).

The KDF separation guarantees the protection of the master key. Besides, the TGF allows the decryption algorithm to operate without computing valid tags for invalid forgery attempts, making it harder for the attacker to use leakage to forge messages. The role of the message block feedback in the tweak inputs is to naturally render the process non-invertible and the internal computation can actually be modelled as an extension of one the hash functions proposed by Black *et al.* [BRS02] under the assumption that Skinny-128/384+ is secure in the chosen-tweakey model. This mode, almost identical to Romulus-N, achieves

the strong ciphertext integrity with misuse and leakage in the chosen-ciphertext model (CIML2) up to the birthday bound. It furthermore achieves integrity nonce-misuse resistance (MR-CINT) and integrity with the release of unverified plaintexts (INT-RUP) up to the birthday bound. Besides, it guarantees the nonce-misuse resilience of messages encrypted with fresh nonces, as long as the challenge queries are leak-free (CCAml1).

To put it into perspective, the model we are targeting is an adversary who has gray-box access to the device for a period of time, where he can perform side-channel attacks with unbounded leakage and force the nonce to be repeated. During this period, there is no rationale for privacy as the adversary can gain information about the plaintexts being encrypted using leakage. However, Romulus-LR ensures that the adversary cannot break the integrity of any message with less than $O(2^{n/2})$ computations. Once the adversary loses access to the device, the security goes back to the black-box level. Hence, a side-channel attack requires birthday-bound complexity and does not harm the long term security of the master key. The details for this new mode are given in [GKP20].

The security of Romulus-LR is similar to modes like Spook [BBB+19] in our targeted use case. While Spook requires computational complexity of $O(2^{n-log(n)})$ computations to break CIML2 security, it requires the use of two different primitives, one of them being a large permutation ($3n \sim 4n$ bits) and the other one is a secure TBC. Romulus-LR requires a single TBC primitive, hence offering a more lightweight trade-off for implementations for both levelled and non-levelled implementations. The difference in security levels is a trade-off that we skewed towards lightweightness. Similarly, Spook covers a slightly stronger model where the encryption leakage also suffer from leakage (CCAmL1). We believe that the trade-off towards lightweightness justifies our choices. In order to address even stronger adversaries, we offer Romulus-LR-TEDT.

The performance of Romulus-LR is almost the same as Romulus-N, except that protected KDF and TGF are required, and proper modifications are done to the tweakey to accommodate the plaintext.

Romulus-LR-TEDT is our second and most advanced leakage resilient mode, directly based on the provably secure TBC-based TEDT construction [BGP+19]. This Romulus-LR-TEDT mode basically consists in fine-tuning the details of TEDT to fit the advantages of Skinny-128/384+ and allow 128-bit nonce and long message/associated data inputs. TEDT provides full leakage resilience, that is, it limits the exploitability of physical leakages via side-channel attacks, even if these leakages happen during every message encryption and decryption operation. TEDT offers what is currently considered as the highest possible security notions in the presence of leakage, namely beyond birthday bound CIML2 and security against Chosen Ciphertext Attacks with nonce-misuse-resilience and Leakage using levelled implementations (CCAmL2). Appendix A of [BGP+19] includes a discussion on why security against chosen ciphertext attacks with nonce-misuse ***resistance*** and leakage is hard to achieve given when the decryption circuit is not leak-free.

While the initial TEDT proposal requires 4 TBC calls to process one n-bit message block, we optimize this to only 3 calls taking advantage of the properties of Skinny and our proposed hash function Romulus-H. This makes the performance more lightweight and closer to typical two-pass SIV-based schemes, which require 2 calls (except Romulus-M which requires only 1.5 calls). Combined with its beyond birthday bound black-box and leakage-resilient security guarantees, it offers a great trade-off for sensitive applications.

Given Romulus-N, Romulus-M, Romulus-LR, Romulus-LR-TEDT, we believe our candidate offers variants that cover the whole spectrum of security levels and use-cases.

5 RUP Security of Romulus-M

Release of unverified plaintext (RUP) is a security notion introduced by Andreeva *et al.* [ABL+14a]. It captures the scenario where the verification can leak the result of decryption (i.e., possibly an unauthentic plaintext) before the verification result is obtained. This is relevant in particular when the verifier's device has a limited amount of memory.

INT-RUP. Among several notions of RUP, authenticity/integrity under RUP, referred to as INT-RUP, is a popular notion for its importance and the possibility to achieve it without heavy constructions, such as the encode-then-encipher approach [BR00] with a wide-block primitive.

It is known that the generic SIV construction achieves INT-RUP security [ABL+14b, Proposition 11] (though there is an error, see below). However, in general, this does not extend to dedicated constructions based on SIV. For instance, an INT-RUP attack is known for SUNDAE [CDD+19].

We prove that Romulus-M is INT-RUP secure even though Romulus-M is not entirely based on SIV, mainly due to the involvement of nonce in the encryption. The proof is fairly straightforward. It shows strong authenticity of Romulus-M under RUP, both against nonce-respecting and nonce-misusing adversaries.

Let $\mathcal{A}$ be the authenticity adversary against AEAD Π. Let $\mathcal{K}$, $\mathcal{N}$, $\mathcal{M}$, $\mathcal{A}$, $\mathcal{T} = \{0,1\}^\tau$ be the key space, the nonce space, the message space, and the τ-bit tag space associated to Π. We assume that Π has three oracles, namely the encryption oracle $\Pi.\mathcal{E} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{M} \times \mathcal{T}$, (unverified) decryption oracle $\Pi.\mathcal{D} : \mathcal{K} \times \mathcal{A} \times \mathcal{M} \to \mathcal{M}$, and verification oracle $\Pi.\mathcal{V} : \mathcal{K} \times \mathcal{A} \times \mathcal{M} \to \{\top, \bot\}$. Here, $\top$ and $\bot$ mean the acceptance and rejection symbols.

We briefly describe INT-RUP security. See [ABL+14a] for details. In the INT-RUP game, $\mathcal{A}$ has access to these three oracles, and if it receives $\top$ from $\Pi.\mathcal{V}$ by making a non-trivial forgery query, then $\mathcal{A}$ is said to (successfully) forge. Here, a forgery query $(N', A', C', T') \in \mathcal{N} \times \mathcal{A} \times \mathcal{M} \times \mathcal{T}$ is said to be non-trivial if there is no previous encryption query (N, A, M) and its response (C, T) that satisfies $(N, A, C, T) = (N', A', C', T')$.

Let

$$\mathbf{Adv}^{\mathtt{auth\text{-}rup}}_{\Pi}(\mathcal{A}) = \Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\Pi.\mathcal{E},\Pi.\mathcal{D},\Pi.\mathcal{V}} \text{ forges}]$$

be the INT-RUP advantage of $\mathcal{A}$ when $\mathcal{A}$ is nonce-respecting for encryption queries. If $\mathcal{A}$ is nonce-misusing, we write $\mathbf{Adv}^{\mathtt{nm\text{-}auth\text{-}rup}}_{\Pi}(\mathcal{A})$ instead.

We write $(q_e, q_d, q_v, t, \sigma)$-adversary to mean an INT-RUP adversary using q_e encryption queries, q_d unverified decryption queries, q_v verification queries, with total time complexity t and the total number of TBC calls σ. We have the following result.

Proposition 1. *Let $\mathcal{A}$ be a nonce-respecting, $(q_e, q_d, q_v, t_A, \sigma_A)$-adversary, and let $\mathcal{B}$ be a nonce-misusing $(q_e, q_d, q_v, t_B, \sigma_B)$-adversary. Then we have*

$$\mathbf{Adv}^{\mathtt{auth\text{-}rup}}_{\mathsf{Romulus\text{-}M}}(\mathcal{A}) \le \mathbf{Adv}^{\mathtt{tprp}}_{\widetilde{E}}(\mathcal{A}') + \frac{5q_v}{2^n},$$
$$\mathbf{Adv}^{\mathtt{nm\text{-}auth\text{-}rup}}_{\mathsf{Romulus\text{-}M}}(\mathcal{B}) \le \mathbf{Adv}^{\mathtt{tprp}}_{\widetilde{E}}(\mathcal{B}') + \frac{4rq_e + 5rq_v}{2^n}$$

for some $\mathcal{A}'$ using σ_A queries with time $t_A + O(\sigma_A)$, and some $\mathcal{B}'$ using σ_B queries with time $t_B + O(\sigma_B)$.

The bounds are essentially the same as the regular authenticity bounds of Romulus-M, i.e., [IKMP20, Theorem 2] for nonce-respecting and [IKMP20, Theorem 3] for nonce-misusing adversaries. This proposition is almost immediate from the proofs of regular authenticity bounds of Romulus-M. In proving the authenticity bounds of Romulus-M based on a TURP $\widetilde{\mathsf{P}}$, we allow the adversary to freely access the encryption/decryption part of Romulus-M (this is possible since the domain separation is explicitly done by the tweaks). This implies that the unverified decryption queries are pointless, since they can be simulated.

An Error in the RUP Analysis of SIV. Proposition 11 of [ABL+14b], which is the full version of [ABL+14a], is not correct in that it reduces the INT-RUP security of generic SIV (named as PRF-to-IV) to the PRF security of the internal MAC function. This is incorrect as the INT-RUP advantage has a non-zero lower bound for any scheme, i.e., by random guessing of a tag, while the PRF advantage can be zero when the MAC function is uniformly random.

Plaintext Awareness. The privacy notion in the RUP setting is called plaintext awareness [ABL+14a]. Intuitively, it requires the existence of an extractor that can simulate the (unverified) decryption oracle without knowing the secret key. It has two versions, called PA1 and PA2, and the stronger notion of PA2 can be achieved only with a wide-block CCA-secure (tweakable) block cipher used in the encode-then-encipher approach (e.g., AEZ [HKR15] or various TESs (Tweakable Enciphering Schemes)). It is easy to see that Romulus-M does not achieve PA2.

It is known that the SIV construction is PA1 secure [ABL$^+$14b, Proposition 6]. The construction of Romulus-M is not covered by the SIV construction (due to the use of a nonce in the encryption part), however, it can be shown that Romulus-M is PA1 secure by following the proof of [ABL$^+$14b, Proposition 6], which proves that the scheme is PA1 secure if the MAC part is a PRF and the encryption part is PA1 secure. The MAC part of Romulus-M is a secure PRF, and the encryption part can be proved to be PA1 secure by following the proof of PA1 security of CBC mode and CTR mode [ABL$^+$14b, Proposition 12].

We start with introducing necessary definitions from [ABL$^+$14a]. Let $\Pi = (\Pi.\mathcal{E}_K, \Pi.\mathcal{D}_K, \Pi.\mathcal{V}_K)$ be an AEAD scheme. Let $\mathcal{A}$ be an adversary with access to two oracles $\mathcal{O}_1$ and $\mathcal{O}_2$. Let $\mathbf{E}$ be an algorithm with access to the history of queries made to $\mathcal{O}_1$ by $\mathcal{A}$, called a PA1-extractor. $\mathbf{E}$ maintains state across invocations. The PA1 advantage of $\mathcal{A}$ relative to $\mathbf{E}$ and Π is

$$\mathbf{Adv}^{\mathrm{PA1}}_{\Pi,\mathbf{E}}(\mathcal{A}) = \Pr[\mathcal{A}^{\Pi.\mathcal{E}_K,\Pi.\mathcal{D}_K} \Rightarrow 1] - \Pr[\mathcal{A}^{\Pi.\mathcal{E}_K,\mathbf{E}} \Rightarrow 1],$$

where the probability is taken over the key K, the randomness of $\mathcal{A}$, and the randomness of $\mathbf{E}$.

Following the PRF-to-IV approach in [ABL$^+$14b, Section 6], we first re-formalize the construction to fit with Romulus-M. Let $G_K : (N, A, M) \mapsto T$ be the MAC part of Romulus-M, and $\mathrm{Enc}_K : (N, M, T) \mapsto C$ be the encryption part. We can re-formalize the oracles $(\Pi.\mathcal{E}_K, \Pi.\mathcal{D}_K, \Pi.\mathcal{V}_K)$ of Romulus-M as follows.

$\Pi.\mathcal{E}_K(N, A, M)$	$\Pi.\mathcal{D}_K(N, C, T)$	$\Pi.\mathcal{V}_K(N, A, C, T)$
$T \leftarrow G_K(N, A, M)$	$M \leftarrow \mathrm{Dec}_K(N, C, T)$	$M \leftarrow \mathrm{Dec}_K(N, C, T)$
$C \leftarrow \mathrm{Enc}_K(N, M, T)$	return M	$T^* \leftarrow G_K(N, A, M)$
return (C, T)		if $T^* = T$ then return M, else return $\perp$

Here, $\mathrm{Dec}_K : (N, C, T) \mapsto M$ is the decryption of Enc_K. We note that although G_K and Enc_K use the same key, they are independent due to the domain separation.

We now define the PA1 advantage of an adversary $\mathcal{A}_1$ relative to $\mathbf{E}$ and Enc for random T as

$$\mathbf{Adv}^{\mathrm{PA1}}_{\mathrm{Enc},\mathbf{E}}(\mathcal{A}_1) = \Pr[\mathcal{A}^{\mathrm{Enc}_K,\mathrm{Dec}_K} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathrm{Enc}_K,\mathbf{E}} \Rightarrow 1],$$

where the probability is taken over the key K and the randomness of $\mathcal{A}_1$, Enc_K, and $\mathbf{E}$. Here, the oracle Enc_K takes (N, M) as input, generates $T \xleftarrow{\$} \{0,1\}^n$, computes $C \leftarrow \mathrm{Enc}_K(N, M, T)$, and returns (C, T). The oracle Dec_K takes (N, C, T) as input and returns $M \leftarrow \mathrm{Dec}_K(N, C, T)$.

We closely follow [ABL$^+$14b, Section 6] to present a proposition showing that if the MAC part G_K of Romulus-M is a secure PRF and the encryption part Enc_K is PA1 secure, then the entire AEAD scheme is PA1 secure.

Proposition 2. *Let* $\mathbf{E}$ *be a PA1 extractor for* Enc *with random* T*. Then there exists an extractor* $\widetilde{\mathbf{E}}$ *for* Π *with arbitrary* N *such that for all adversaries* $\mathcal{A}$ *there exist* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *such that*

$$\mathbf{Adv}^{\mathrm{PA1}}_{\Pi,\widetilde{\mathbf{E}}}(\mathcal{A}) \le \mathbf{Adv}^{\mathrm{PA1}}_{\mathrm{Enc},\mathbf{E}}(\mathcal{A}_1) + \mathbf{Adv}^{\mathrm{prf}}_{G}(\mathcal{A}_2),$$

where $\mathbf{Adv}^{\mathrm{prf}}_{G}(\mathcal{A}_2)$ *is the prf advantage of* $\mathcal{A}_2$ *relative to* G*.*

We note that we are considering arbitrary nonces on Π, and adversaries are free to repeat nonces. However, we assume that the adversary does not repeat the same query.

Proof. We define $\widetilde{\mathbf{E}}$ as follows. First, $\widetilde{\mathbf{E}}$ transforms its query history by mapping $\mathcal{E}_K(N, A, M) = (C, T)$ to $\mathrm{Enc}_K(N, M, T) = C$. Then on input (N, C, T), $\widetilde{\mathbf{E}}$ returns $\mathbf{E}(N, C, T)$.

Now let $\widetilde{\mathcal{O}}_1$ and $\widetilde{\mathcal{O}}_2$ be the oracles of $\mathcal{A}$, and $\mathcal{O}_1$ and $\mathcal{O}_2$ be the oracles of $\mathcal{A}_1$. The adversary $\mathcal{A}_1$ runs $\mathcal{A}$, and on a query $\widetilde{\mathcal{O}}_1(N, A, M)$ made by $\mathcal{A}$, $\mathcal{A}_1$ responds with $(C, T) \leftarrow \mathcal{O}_1(N, M)$ if (N, M) was not previously queried, otherwise it responds with the previous (C, T). On a query $\widetilde{\mathcal{O}}_2(N, C, T)$ made by $\mathcal{A}$, $\mathcal{A}_1$ responds with $M \leftarrow \mathcal{O}_2(N, C, T)$. We see that $\mathcal{A}_1$ perfectly simulates the PA1 game for $\mathcal{A}$ as long as G is a PRF. □

As G is a PRF [IKMP20, Lemma 7], it remains to show that the encryption part is PA1 secure.

We recall how Enc_K works. Let K be a key, and let $E_K : (N, i, S) \mapsto U$ be a tweakable block cipher, where we write $E_K^{N,i}(S) = U$. We also let $\rho : (S, M) \mapsto (S', C)$ be the state update function, where $C = M \oplus G(S)$, $S' = S \oplus M$, and $G(S)$ is a multiplication of S and a fixed matrix G. Let (N, M, T) be the input of the encryption algorithm Enc_K, where $M = (M[1], \ldots, M[m])$, $|M[i]| = n$ for $1 \le i \le m-1$, and $0 \le |M[m]| \le n$. Then the output of $\mathrm{Enc}_K(N, M, T)$ is C, where $C = (C[1], \ldots, C[m])$ and for $0 \le i \le m-2$, we let $U[i] \leftarrow E_K^{N,i}(S[i])$ and $(S[i+1], C[i+1]) \leftarrow \rho(U[i], M[i+1])$. Here, we let $S[0] = T$ and for $i = m-1$, we let $U[m-1] \leftarrow E_K^{N,m-1}(S[m-1])$ and $C[m] \leftarrow \mathrm{lsb}_{|M[m]|}(U[m-1]) \oplus M[m]$.

Now let $\mathcal{A}_1$ be a PA1 adversary that makes q_e encryption queries that consist of σ_e message blocks in total and q_d decryption queries that consist of σ_d ciphertext blocks in total, where $q_e + q_d \le q$ and $\sigma_e + \sigma_d \le \sigma$. Following the analyses of CBC mode and CTR mode in [ABL+14b], in our security analysis to derive the upper bound on $\mathbf{Adv}^{\mathrm{PA1}}_{\mathrm{Enc},\mathbf{E}}(\mathcal{A}_1)$, we only consider a simplified case where the oracle Enc_K takes as input only (N, M) such that $|M| = mn$ for some integer $m \ge 0$, and the oracle Dec_K takes as input only (N, C, T) such that $|C| = mn$ for some integer $m \ge 0$. We also use a pseudorandom function $F_K : (N, i, S) \mapsto U$ as E_K, which is regarded as a random function at the cost of $\mathbf{Adv}^{\mathrm{tprp}}_E(\mathcal{A}')$ for some $\mathcal{A}'$ plus $\sigma^2/2^n$ terms in the PA1 advantage.

We define an extractor $\mathbf{E}$ as follows. First, $\mathbf{E}$ chooses a key K' which will be used in a pseudorandom function $F_{K'}$. The extractor maintains two lists, $\mathcal{L}$ and $\mathcal{L}'$, to keep the record of the input-output pairs of F_K and $F_{K'}$. These lists are initially empty, and works as follows. Let (N_i, M_i, T_i, C_i) be the i-th Enc_K query-response pair of $\mathcal{A}$, where M_i (and C_i) has m_i blocks. Let $((N_i, 0, S_i[0]), U_i[0]), \ldots, ((N_i, m_i - 1, S_i[m_i - 1]), U_i[m_i - 1])$ be the list of input-output pairs of F_K. The adversary records them into $\mathcal{L}$.

Let (N_j^*, C_j^*, T_j^*) be the j-th decryption query of $\mathcal{A}$ given to $\mathbf{E}$, where $C_j^* = (C_j^*[1], \ldots, C_j^*[m_j^*])$ has m_j^* blocks. The approach is to use $\mathcal{L}$ whenever possible, otherwise we use $F_{K'}$, and we make use of $\mathcal{L}'$ to maintain consistency to previously decrypted messages.

In more detail, to decrypt the k-th block $C_j^*[k]$, we need the value of $U_j^*[k-1] = F_K^{N_j^*,k-1}(S_j^*[k-1])$. If the input $(N_j^*, k-1, S_j^*[k-1])$ is in $\mathcal{L}$, then we let the corresponding output as $U_j^*[k-1]$. Otherwise, if the input $(N_j^*, k-1, S_j^*[k-1])$ is in $\mathcal{L}'$, then we let the corresponding output as $U_j^*[k-1]$. Otherwise, if $(N_j^*, k-1, S_j^*[k-1])$ is not included in $\mathcal{L}$ nor $\mathcal{L}'$, then we choose $U_j^*[k-1]$ uniformly at random from $\{0,1\}^n$, and record $((N_j^*, k-1, S_j^*[k-1]), U_j^*[k-1])$ into $\mathcal{L}'$ as the input-output pair of $F_{K'}$. Now with the value of $U_j^*[k-1]$, the k-th ciphertext block $C_j^*[k]$ is decrypted into $M_j^*[k] = G(U_j^*[k-1]) \oplus C_j^*[k]$, and the extractor returns $M_j^* = (M_j^*[1], \ldots, M_j^*[m_j^*])$ to $\mathcal{A}$.

The simulation can fail. Assume that $\mathcal{A}$ makes the i-th encryption query (N_i, M_i), then T_i is randomly chosen, and let $(N_i, 0, S_i[0]), (N_i, 1, S_i[1]), \ldots$ be the input values of F_K to compute C_i. The simulation fails if some input $(N_i, k, S_i[k])$ was previously used in a decryption query through $F_{K'}$, in which case, we have an obvious contradiction and the simulation fails. Otherwise, the consistency to all the previously returned values is maintained, and we see that the simulation works.

Next, we extend this and let the simulation fail (equivalently, let the adversary win the distinguishing game) if some input $(N_i, k, S_i[k])$ was used as an input value of F_K or $F_{K'}$ in any of the previous encryption queries or decryption queries. That is, we require that all the input values $(N_i, 0, S_i[0]), (N_i, 1, S_i[1]), \ldots$ are not included in $\mathcal{L}$ nor $\mathcal{L}'$, and all the output values $U_i[0], U_i[1], \ldots$ are randomly chosen at this i-th encryption query. We see that the simulation succeeds if this is the case.

Let m_i be the number of blocks of M_i. Then the probability that $\mathcal{A}$ wins with the i-th encryption query is at most $qm_i/2^n$, since for each $(N_i, k, S_i[k])$, we have at most q target values that make $\mathcal{A}$ win, and we have m_i blocks to consider. Therefore, the PA1 advantage of $\mathcal{A}_1$ relative to Enc with a PRF F_K and $\mathbf{E}$ is at most $q(m_1 + \cdots + m_{q_e})/2^n \le \sigma^2/2^n$, and this shows that the encryption part of Romulus-M is PA1 secure up to the birthday bound.

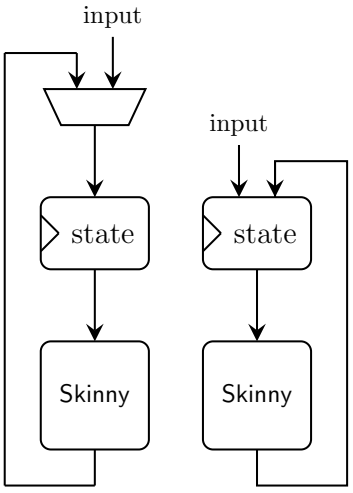
6 New Hardware and Threshold Implementations of Romulus

In this section, we provide a full list of different ASIC implementation trade-offs of Romulus-N1, showing the design range of our proposal. We will study round-based implementations, serial implementations and threshold implementations. Note that unless specified otherwise the performance numbers reported here are for Romulus-N1 and not Romulus-N. Thus, in order to get a good estimation of Romulus-N performance, one can simply apply a 1.4 factor to the throughput numbers displayed.

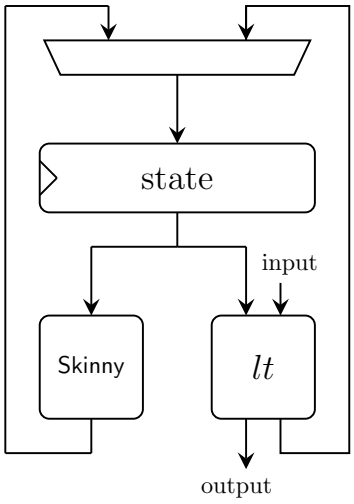
6.1 Round-Based Architecture

The goal of the design of Romulus is to have a very small area overhead over the underlying TBC, specially for the round-based implementations. In order to achieve this goal, we set two requirements:

1. There should be no extra Flip-Flops over what is already required by the TBC, since Flip-Flops are very costly ($4 \sim 7$ GEs per Flip-Flop).
2. The number of possible inputs to each Flip-Flop and outputs of the circuits have to be minimized. This is in order to reduce the number of multiplexers required, which is usually one of the causes of efficiency reduction between the specification and implementation.



(a) Overview of the round based architecture of Skinny.



(b) Overview of the round based architecture of Romulus. *lt* represents the linear transformation that includes ρ, the block counter and the inverse key schedule.

Fig. 3: Expected architectures for Skinny and Romulus

One of the advantages of Skinny as a lightweight TBC is that it has a very simple datapath, consisting of a simple state register followed by a low-area combinational circuit, where the same circuit is used for all the rounds. Thus, the only multiplexer required is to select between the initial input for the first round and the round output afterwards (Figure 3(a)) and it has been shown that this multiplexer can even have lower cost than a normal multiplexer if combined with the Flip-Flops by using Scan-Flops (Figure 3(b)) [JMPS17]. However, when used inside an AEAD mode, challenges arise. For example, how to store the key and nonce, as the key scheduling algorithm will change these values after each block encryption. The same goes for the block counter. In order to avoid duplicating the storage elements for these values (one set to be used to execute the TBC and one set to be used by the mode to maintain the current value), we studied the relation between the

original and final value of the tweakey through one TBC call. Since the key scheduling algorithm of Skinny is fully linear and requires a very low area to be implemented (most of the algorithm is just routing and renaming of different bytes), the full algorithm can be inverted using a very small circuit that costs 64 XOR gates for Romulus-N1. Moreover, the Linear Feedback Shift Register (LFSR) computation required between blocks can be implemented on top of this circuit, costing 3 extra XOR gates. This operation can be computed in parallel to ρ, such that when the state is updated for the next block, the tweakey required is also ready. This costs only ~ 67 XOR gates as opposed to ~ 320 Flip-Flops that would, otherwise, be needed to maintain the tweakey value. Hence, the mode was designed with the architecture in Figure 3(b) in mind, where only a full-width state-register is used, carrying the TBC state and tweakey values naturally. Every cycle, it is either kept without change, updated with the TBC round output (which includes a single round of the key scheduling algorithm) or the output of a simple linear transformation, which consists of ρ/ρ^{-1}, the unrolled inverse key schedule and the block counter. In order to estimate the hardware cost of Romulus-N1, we consider the round-based implementation with an $n/4$-bit input/output bus:

- 4 XOR gates for computing G.
- 64 XOR gates for computing ρ.
- 67 XOR gates for the correction of the tweakey and counting.
- 56 multiplexers to select whether to choose to increment the counter or not.
- 320 multiplexers to select between the output of the Skinny round and lt.

This adds up to 135 XOR gates and 376 multiplexers. For estimation purposes, assuming an XOR gate costs 2.25 GEs and a multiplexer costs 2.75 GEs, this adds up to 1337.75 GEs. In the original Skinny paper [BJK+16], the authors reported that Skinny-128/384 requires $4,268$ GEs (we assume Skinny-128/384+ takes the same area since only the number of rounds is reduced), which adds up to $\sim 5,605$ GEs. This is for example ~ 1.4 KGEs smaller than the round-based implementation of Ascon [GWDE15]. Moreover, a smarter design can make use of the fact that 64 bits of the tweakey of Skinny-128/384+ are not used, replacing 64 Flip-Flops by 64 multiplexers reducing the area by an extra ~ 200 GEs. In order to design a combined encryption/decryption circuit, we show below that the decryption costs only an extra 32 multiplexers and ~ 32 OR gates, which is equivalent to ~ 100 GEs.

These estimations show that Romulus-N is not just competitive theoretically but it can be a very attractive option practically for low area applications. To put into perspective, the 8-bit implementation of ACORN (one of the two selected schemes in the "Lightweight applications" portfolio of the competition), the smallest implementation among the round 3 candidates of the CAESAR competition that is compliant with the benchmarking API, costs $5,900$ GEs, as shown in [KHYKC17].

Another possible optimization is to consider the fact that most of the area of Skinny comes from the storage elements. Hence, we can speed up Romulus to almost double the speed by using a simple two-round unrolling, which costs $\sim 1,000$ GEs, as only the logic part of Skinny needs replication, which is only $< 20\%$ increase in terms of area. First we study the impact of different number of round-unrolling, where `Rx` means round-based implementation with `x` round unrolling. We used Synopsys Design Compiler and the TSMC 65nm technology library for our measurements. We varied the operating frequency from the maximum possible till 125 MHz and see the impact on throughput, area, power and energy. The results are given in Table 2.

Romulus-M is estimated to have almost the same area as Romulus-N, except for an additional set of multiplexers in order to use the tag as initial vector for the encryption part. This indicates that it can be a very lightweight choice for high security applications.

6.2 Serial Implementations

With regards to serial implementations, we followed the currently popular bit-sliding framework [JMPS17] with minor tweaks. The state of Skinny is represented as a Feedback-Shift Register which typically operates on 8 bits at a time, while allowing the 32-bit MixColumns operation, given in Figure 4.

Table 2: The design space of Romulus-N1 using the TSMC-65 ASIC technology: (unrolled) round-based implementations. `Rx` means round-based implementation with `x` round unrolling.

Arch.	Critical Path (ns)	Area (GE)	Power (mW)	Energy (Enc block) (pJ)	Energy (Auth block) (pJ)	Throughput (Enc only) (Mbps)	Throughput (Auth only) (Mbps)	Throughput ($\|A\| = \|M\|$) (Mbps)	Thput/Area
R1	0.5	6220	0.8	24	12.8	4267	8000	6133	0.98
R1	0.66	5877	0.72	28.5	15.2	3232	6060	4647	0.79
R1	0.75	5864	0.65	29.25	15.6	2844	5333	4089	0.69
R1	1	5860	0.53	31.8	16.9	2133	4000	3067	0.52
R1	2	5860	0.35	42	22.4	1067	2000	1533	0.26
R1	4	5860	0.25	60	32	533	1000	767	0.13
R1	8	5772	0.22	105.6	56.3	266	500	383	0.07
R2	0.5	7978	0.99	15.84	8.91	8000	14200	11100	1.39
R2	0.66	7161	0.78	16.47	9.2	6060	10758	8410	1.17
R2	0.75	6860	0.71	17.04	9.5	5333	9467	7400	1.07
R2	1	6727	0.56	17.92	10.08	4000	7100	5550	0.82
R2	2	6635	0.38	24.32	13.68	2000	3550	2775	0.41
R2	4	6635	0.29	60	20.88	1000	1775	1388	0.2
R2	8	6635	0.25	64	36	500	888	694	0.1
R4	0.8	12766	1.1	15.84	9.68	8888	14500	11694	0.91
R4	1	10008	0.77	13.86	8.47	7111	11600	9356	0.93
R4	2	8740	0.46	16.56	10.12	3555	5800	4678	0.53
R4	4	8740	0.36	25.92	15.84	1775	2900	2339	0.26
R4	8	8740	0.32	46.08	28.16	889	1450	1170	0.13
R8	1.6	18679	1.2	21.12	14.4	7250	10662	8956	0.48
R8	2	15133	0.89	19.58	13.35	5800	8530	7165	0.47
R8	4	12990	0.51	22.44	15.3	2900	4265	3582	0.28
R8	8	12990	0.45	39.6	27	1450	2132	1791	0.14

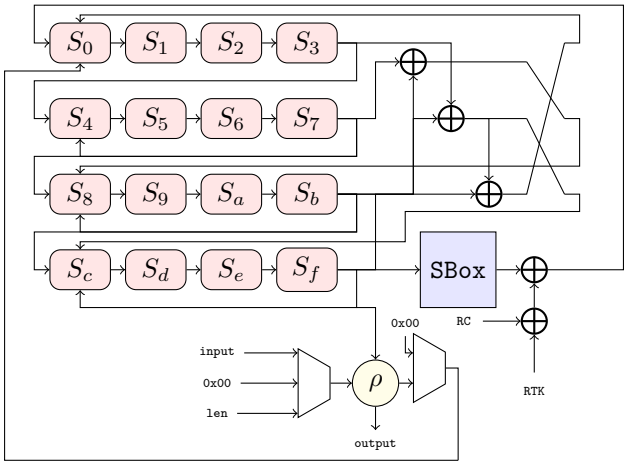


Fig. 4: Serial state update function used in Romulus.

It can be viewed in Figure 4 that several careful design choices such as a lightweight serializable ρ function without the need of any extra storage and a lightweight padding/truncation scheme allow the low area implementations to use a very small number of multiplexers on top of the Skinny circuit for the state update (three 8-bit multiplexers to be exact, two of which have a constant zero input, and ~ 22 XORs for the ρ function and block counter). Regarding the key update functions, we conducted several experiments on how to serialize the operations and we found that the best trade-off is to design a parallel/serial register for every tweakey, where the key schedule and mode operations are done in the same manner as the round-based

implementation, while the AddRoundKey operation of Skinny is done serially as shown in Figure 4. In Table 3, we report measurements for the byte serial architecture based on the bit-sliding technique, following the same methodology.

Table 3: The design space of Romulus-N1 using the TSMC-65 ASIC technology: byte serial implementations

Arch.	Critical Path (ns)	Area (GE)	Power (mW)	Energy (Enc block) (pJ)	Energy (Auth block) (pJ)	Throughput (Enc only) (Mbps)	Throughput (Auth only) (Mbps)	Throughput ($\|A\| = \|M\|$) (Mbps)	Thput/Area
S1	0.75	3390	0.5	489	247.5	131	259	195	0.06
S1	1	3318	0.5	652	330	98	194	146	0.04
S1	2	3318	0.29	756	383	49	97	73	0.02
S1	4	3318	0.2	1043	528	25	49	37	0.01
S1	8	3318	0.15	1565	792	12	24	18	0.005

6.3 Threshold Implementations

Table 4: The design space of Romulus-N1 using the TSMC-65 ASIC technology: byte serial implementations and round based first-order threshold implementations

Arch.	Critical Path (ns)	Area (GE)	Power (mW)	Energy (Enc block) (pJ)	Energy (Auth block) (pJ)	Throughput (Enc only) (Mbps)	Throughput (Auth only) (Mbps)	Throughput ($\|A\| = \|M\|$) (Mbps)	Thput/Area
PS	0.75	5163	0.79	772	391	131	259	195	0.04
PS	1	5158	0.62	808	409	98	194	146	0.03
PS	2	5154	0.40	1043	529	49	97	73	0.01
PS	4	5154	0.27	1408	713	25	49	37	0.007
PS	8	5154	0.21	2190	1110	12	24	18	0.003
P1	0.5	8386	1.2	36	19.2	4267	8000	6133	0.73
P1	0.66	8101	0.89	35.2	18.7	3232	6060	4647	0.57
P1	0.75	8048	0.8	36	19.2	2844	5333	4089	0.51
P1	1	8048	0.69	41.4	22.1	2133	4000	3067	0.38
P1	2	8048	0.46	55.2	29.4	1067	2000	1533	0.19
P1	4	8048	0.35	84	44.8	533	1000	767	0.095
P1	8	8048	0.28	134.4	51.6	266	500	383	0.05

In Table 4, we study the 3-share threshold implementation of both the byte serial architecture (PS) and the single round architecture (P1). The implementations are based on the threshold implementations provided by the Skinny team [BJK+16]. As can be seen in the comparison figure below, our threshold implementations are very competitive. This partially comes from the fact that high level security can be achieved with a small state when using TBCs, which leads to smaller overhead when protecting implementations against side-channel attacks, as already pointed out by [NSS20a] and used in Spook candidate [BBB+19]. For example, while the gap between Romulus-N1 and Ascon may not be huge for unprotected implementations, the gap is big when it comes to threshold implementations, as shown in Figure 5.

Summary of hardware implementations. Our results indicate that the best throughput/area trade-off is achieved by the R2 architecture, which processes two rounds of the TBC per cycle. The minimum energy is achieved by 4-round unrolling (the R4 architecture), as it requires only 18 cycles per block for encryption and 11 cycles per block for associated data. The serial implementation can be as low as 3.3 KGE, which is extremely low for an AEAD mode with full n-bit security and standard model security proofs. On the other hand, for about 8 KGE, we can have a very efficient threshold implementation protected against side-channel attacks.

7 Hardware/Software Performance Comparison

Hardware. In Figure 5, we compare the design space of Romulus-N1 and Romulus-N to the winners of the CAESAR competition for "Lightweight applications" portfolio, namely Ascon [DEMS16] and ACORN [Wu16] (as winners, they can be considered as the top of the state-of-the-art).

Romulus-N1 and Romulus-N offer a competitive performance compared to Ascon and ACORN for unprotected implementations, while it offers particularly excellent performance for threshold implementations. Compared to Ascon, this comes from the fact (as explained in [NSS20a]) that while sponge-based constructions use a large permutation with a lot of non-linear operations, TBC-based schemes use a smaller permutation with cheaper and usually fully-linear key scheduling algorithms. This means that protecting the key scheduling algorithm is both cheaper and less demanding [BJK+, NSS20b, NSGD12]. Besides, our new variant Romulus-N achieves ∼15 Gbps throughput with only about 8 KGE, which makes it an excellent candidate for high throughput and energy efficient architectures. On the other end of the spectrum, it can reach close to 300 Mbps for ∼3.4 KGE, making it an excellent trade-off for low area and low power applications as well. For threshold implementations, our single round implementation reaches ∼8.5 Gbps using only ∼8.4 KGE. This throughput/area ratio is competitive with even unprotected implementations of Ascon and ACORN. It can also be seen that the low area protected implementations of Romulus-N1 and Romulus-N are very close in area to Ascon's unprotected low-area implementation and an order of magnitude faster.

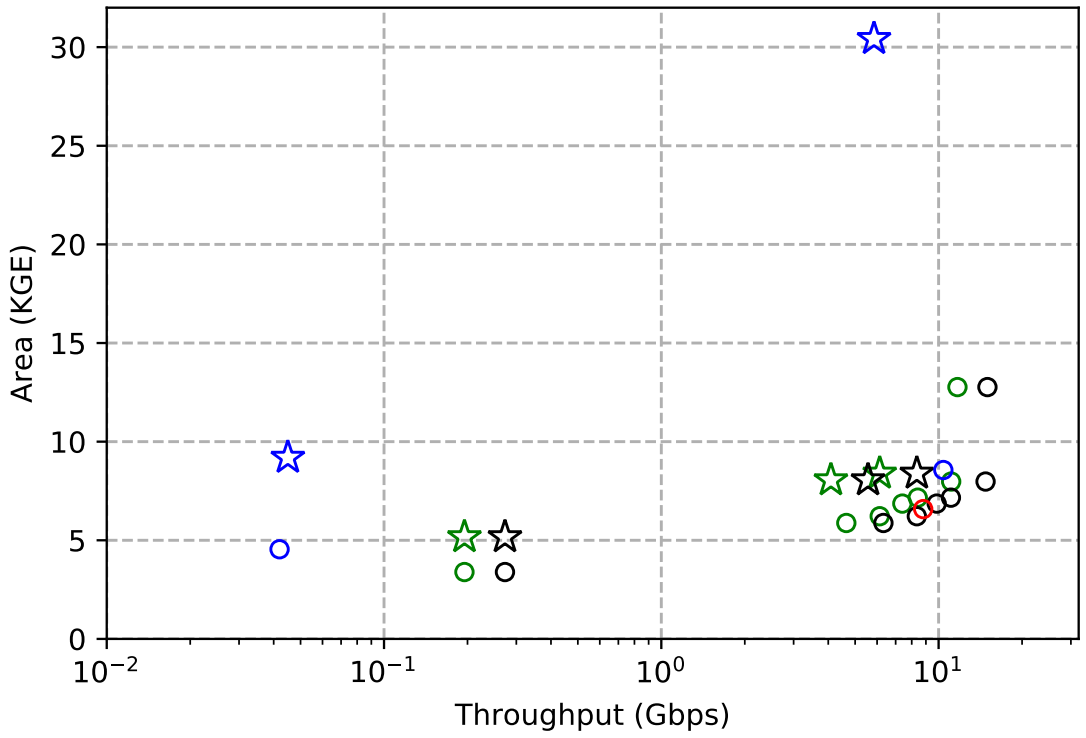


Fig. 5: Throughput vs. Area trade-offs for different schemes. Black: Romulus-N, Green: Romulus-N1, Red: ACORN, Blue: Ascon. ○: unprotected implemetation, ☆: threshold implementation.

Software. Regarding software implementations, we applied the new fixslicing strategy [ANP20] to Skinny which led to good performance results. Referring to the benchmarks from `https://lwc.las3.de/table.php`, we observe that for 32-bit platforms Romulus-N1 is generally placed in the middle of the rankings regarding

throughput (Romulus-N being ranked in the first half). We remark that on these platforms, AES is already performing quite well. However, more interestingly, for very constrained platforms such as 8-bit architectures, Romulus-N1 ranks in the top tier, while Romulus-N would be among the top candidates (applying the 1.4 improvement ratio due to the reduction of the number of rounds). We believe this very constrained platforms (4-bit or 8-bit architectures) are probably the use-cases where lightweight cryptography makes the most sense in software.

Acknowledgements

We would like to thank Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha and Kan Yasuda for their feedback and constructive comments on the error in [ABL+14b]. We thank Yusuke Naito for his feedback on MDPH hash function. We would also like to thank Shivam Bhasin for discussions about the threshold implementations of tweakable block ciphers. The second and fourth authors are supported by Temasek Laboratories, Singapore.

References

ABL+14a. Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to Securely Release Unverified Plaintext in Authenticated Encryption. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 105–125. Springer, 2014.

ABL+14b. Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to Securely Release Unverified Plaintext in Authenticated Encryption. *IACR Cryptology ePrint Archive*, 2014:144, 2014.

AMP10. Elena Andreeva, Bart Mennink, and Bart Preneel. Security Reductions of the Second Round SHA-3 Candidates. In *ISC*, volume 6531 of *Lecture Notes in Computer Science*, pages 39–53. Springer, 2010.

ANP20. Alexandre Adomnicai, Zakaria Najm, and Thomas Peyrin. Fixslicing: A New GIFT Representation Fast Constant-Time Implementations of GIFT and GIFT-COFB on ARM Cortex-M. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):402–427, 2020.

BBB+19. Davide Bellizia, Francesco Berti, Olivier Bronchain, Gaétan Cassiers, Sébastien Duval, Chun Guo, Gregor Leander, Gaétan Leurent, Itamar Levi, Charles Momin, , Olivier Pereira, Thomas Peters, François-Xavier Standaert, and Friedrich Wiemer. Spook v1. Submission to NIST Lightweight Cryptography Project, 2019.

BGP+19. Francesco Berti, Chun Guo, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Tedt, a leakage-resist aead mode for high physical security applications. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):256–320, Nov. 2019.

BJK+. Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The Skinny Cipher Website.

BJK+16. Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In *CRYPTO (2)*, volume 9815 of *Lecture Notes in Computer Science*, pages 123–153. Springer, 2016.

BJK+19. Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. SKINNY-AEAD and SKINNY-Hash. Submission to NIST Lightweight Cryptography Project, 2019.

BR00. Mihir Bellare and Phillip Rogaway. Encode-Then-Encipher Encryption: How to Exploit Nonces or Redundancy in Plaintexts for Efficient Cryptography. In *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 317–330. Springer, 2000.

BRS02. John Black, Phillip Rogaway, and Thomas Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2002.

CDD+19. Donghoon Chang, Nilanjan Datta, Avijit Dutta, Bart Mennink, Mridul Nandi, Somitra Sanadhya, and Ferdinand Sibleyras. Release of Unverified Plaintext: Tight Unified Model and Application to ANYDAE. Cryptology ePrint Archive, Report 2019/1326, 2019. `https://eprint.iacr.org/2019/1326`.

CDMP05. Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård Revisited: How to Construct a Hash Function. In *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448. Springer, 2005.

CSSH19. Qiu Chen, Danping Shi, Siwei Sun, and Lei Hu. Automatic Demirci-Selçuk Meet-in-the-Middle Attack on SKINNY with Key-Bridging. In Jianying Zhou, Xiapu Luo, Qingni Shen, and Zhen Xu, editors, *Information and Communications Security - 21st International Conference, ICICS 2019, Beijing, China, December 15-17, 2019, Revised Selected Papers*, volume 11999 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2019.

DEMS16. Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. Submission to Round 3 of the CAESAR competition, 2016.

GKP20. Chun Guo, Mustafa Khairallah, and Thomas Peyrin. AET-LR: Rate-1 Leakage-Resilient AEAD based on the Romulus Family. Submission to NIST Lightweight Cryptography Workshop, 2020.

GWDE15. Hannes Groß, Erich Wenger, Christoph Dobraunig, and Christoph Ehrenhöfer. Suit up!–Made-to-Measure Hardware Implementations of ASCON. In *2015 Euromicro Conference on Digital System Design*, pages 645–652. IEEE, 2015.

Hir06. Shoichi Hirose. Some plausible constructions of double-block-length hash functions. In *International Workshop on Fast Software Encryption*, pages 210–225. Springer, 2006.

HKR15. Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. Robust authenticated-encryption AEZ and the problem that it solves. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 15–44. Springer, 2015.

HPY12. Shoichi Hirose, Je Hong Park, and Aaram Yun. A Simple Variant of the Merkle-Damgård Scheme with a Permutation. *J. Cryptology*, 25(2):271–309, 2012.

IKMP19. Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. Romulus v1. Submission to NIST Lightweight Cryptography Project, 2019.

IKMP20. Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. Duel of the Titans: The Romulus and Remus Families of Lightweight AEAD Algorithms. *IACR Transactions on Symmetric Cryptology*, (1), 2020. `https://eprint.iacr.org/2019/992`.

JMPS17. Jérémy Jean, Amir Moradi, Thomas Peyrin, and Pascal Sasdrich. Bit-Sliding: A Generic Technique for Bit-Serial Implementations of SPN-based Primitives - Applications to AES, PRESENT and SKINNY. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 687–707. Springer, 2017.

KHYKC17. Sachin Kumar, Jawad Haj-Yihia, Mustafa Khairallah, and Anupam Chattopadhyay. A Comprehensive Performance Analysis of Hardware Implementations of CAESAR Candidates. *IACR Cryptology ePrint Archive*, 2017:1261, 2017.

MRH04. Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2004.

Nai19. Yusuke Naito. Optimally Indifferentiable Double-Block-Length Hashing Without Post-processing and with Support for Longer Key Than Single Block. In *International Conference on Cryptology and Information Security in Latin America*, pages 65–85. Springer, 2019.

NSGD12. M. Nassar, Y. Souissi, S. Guilley, and J. Danger. Rsm: A small and fast countermeasure for aes, secure against 1st and 2nd-order zero-offset scas. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1173–1178, March 2012.

NSS20a. Yusuke Naito, Yu Sasaki, and Takeshi Sugawara. Lightweight Authenticated Encryption Mode Suitable for Threshold Implementation. In *39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings*, volume 12105 of *Lecture Notes in Computer Science*. Springer, 2020.

NSS20b. Yusuke Naito, Yu Sasaki, and Takeshi Sugawara. Lightweight Authenticated Encryption Mode Suitable for Threshold Implementation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques - EUROCRYPT*, Lecture Notes in Computer Science. Springer, 2020.

Rt20. Romulus and SKINNY-AEAD teams. New Romulus and SKINNY-AEAD variants. Announcement to the NIST lwc forum mailing list, May 13, 2020.

SSD+18. Danping Shi, Siwei Sun, Patrick Derbez, Yosuke Todo, Bing Sun, and Lei Hu. Programming the Demirci-Selçuk Meet-in-the-Middle Attack with Constraints. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 3–34. Springer, 2018.

TAY17. Mohamed Tolba, Ahmed Abdelkhalek, and Amr M. Youssef. Impossible differential cryptanalysis of reduced-round SKINNY. In *AFRICACRYPT*, volume 10239 of *Lecture Notes in Computer Science*, pages 117–134, 2017.

Wu16. Hongjun Wu. ACORN: A Lightweight Authenticated Cipher v3. Submission to Round 3 of the CAESAR competition, 2016.

ZDM+19. Boxin Zhao, Xiaoyang Dong, Willi Meier, Keting Jia, and Gaoli Wang. Generalized Related-Key Rectangle Attacks on Block Ciphers with Linear Key Schedule: Applications to SKINNY and GIFT. Cryptology ePrint Archive, Report 2019/714, 2019. https://eprint.iacr.org/2019/714.