HARDWARE IMPLEMENTATIONS OF ROMULUS: EXPLORING NONCE MISUSE RESISTANCE AND BOOLEAN MASKING

MUSTAFA KHAIRALLAH

SHIVAM BHASIN

Nanyang Technological University, Singapore

1. Introduction

In this paper, we investigate the hardware implementation of SKINNY and Romulus. In Section 2, we explore the implementation of Romulus-M, the nonce misuse-resistant variant of Romulus. In Section 3, we explore the first-order masked implementations of the SKINNY SBox. While we focus on the SKINNY SBox, our results and observations apply to any SBox, as they represent underlying issues in the masking schemes.

2. Nonce Misuse: On the Cost of Implementing Romulus-M

Lightweight cryptography is the field of cryptology that studies cryptographic schemes targeted at achieving competitive performance at low cost. Specifically, it targets environments where classical symmetric-key encryption algorithms, such as the Advanced Encryption Standard (AES) [DR01] and the AES-GCM Authenticated Encryption with Associated Data (AEAD) mode [MV04], are either infeasible or too costly. An AEAD mode is a symmetric-key cryptographic scheme designed to offer both integrity and confidentiality, simultaneously. It takes three inputs: plaintext M, secret key K and public associated data A, and outputs a ciphertext C and an authenticated tag T. In 2019, the National Institute for Standardization and Technology (NIST), USA, accepted 56 AEAD candidates for standardization [TMÇ+19], and narrowed them down to 10 finalists in 2021 [TMC+21]. Each submission includes one main variant, and optionally other variants that offer extra features. All the main partially derive their security from an extra public input known as either *nonce* (N) or *Initial Vector* (IV), where for a given secret key K, N must be unique for every message and must not be repeated. While this leads to the design of fast and cheap AEAD scheme, it leads to implementation problems, as the either the implementation, the higher-level communication protocol or the user must ensure that the nonce is never repeated. This requires additional storage and implementation mechanisms not accounted for in the AEAD cost. The additional storage and maintenance is required for every communication channel. Besides, the ambiguity about the responsibility of generating the nonce has been identified by researchers as a security threat. In 2022, Shakevsky *et al.* [SRW22] have shown that Samsung flagship smartphones, including S21, use AES-GCM as an AEAD scheme and are vulnerable to nonce-misuse attacks.

On the flip side, AEAD schemes that are not vulnerable to nonce-misuse attacks, and can be implemented even without any nonce or with a fixed nonce have been proposed

E-mail addresses: `mustafa.khairallah@ntu.edu.sg, sbhasin@ntu.edu.sg`.

by researchers. In their seminal work [RS06], Rogaway and Shrimpton proposed the Synthetic-IV (SIV) scheme, which was followed up by other misuse-resistant schemes ([JNPS16, GLL17]). However, these schemes require the input to be processed twice, which is a limitation that cannot avoided to achieve nonce-misuse.

In the context of the NIST lightweight cryptography standardization project, only Romulus [IKMP] includes a variant; Romulus-M, that offers integrity and confidentiality against nonce-misuse adversaries, while Elephant [BCDM21] only offers integrity against nonce-misuse adversaries, without privacy. However, the hardware performance of Romulus-M has been under-studied.

Contributions. In this paper, the design of a configurable hardware accelerator for the Romulus AEAD family is proposed. The design simultaneously supports the nonce-respecting variant Romulus-N and the misuse-resistant variant Romulus-M. Besides, it can be configured to support the newly proposed ISO standard: ISO/IEC 18033-7 for Tweakable Block Ciphers [Tec21]. We show that Romulus-M can be implemented with almost zero-overhead in terms of area, compared to Romulus-N and less than 60% slow-down.

Comparative studies of new standardization proposals, such as this one, help understand the performance of new designs and inform standards' authors and implementation designers alike. Particularly, understanding the performance of the misuse-resistant AEAD scheme Romulus-M is needed as part of the NIST standardization efforts.

2.1. **Background.**

2.1.1. *Nonces.* The security of SKE is dependent, by definition, on the secrecy of a shared secret key K between communicating parties. In practice, however, the security of many SKE schemes, including most AEAD schemes, relies also on the properties of another parameter that is usually public, and is referred to as a Nonce N or an Initial vector IV. This public parameter is usually assumed to be unique and cannot be repeated for two different messages (nonce-based AEAD) or uniformly random (IV-based AEAD).

Both of the aforementioned requirements are hard to enforce in practice. They rely on a threat model that assumes the user and implementer are both honest and understand the security requirements. To the contrary, it was shown that these assumptions may not hold even in widely-used commercial products, designed with the promise of hardware security. In February 2022, Shakevsky *et al.* showed an IV-reuse attack on the AES-GCM encryption algorithm on a range of flagship Android-powered Samsung smartphones [SRW22].

Such attacks necessitate more careful handling of nonces. This can be done in one of two ways.

(1) Ensure the implementations use nonces that satisfy the assumptions that the AEAD scheme is based on, *e.g.* uniqueness. However, this approach is what is being assumed currently. This leads either complicated and costly implementations, or implementation mistakes that can leads to cryptographic breaks.
(2) Rely on AEAD schemes that are more robust to nonce repetition or non-randomness. On the bright side, this issue have been studied by cryptographers, schemes such as SIV [RS06], AES-GCM-SIV [GLL17], Deoxys-II [JNPS16] and, more recently, Romulus-M.

2.1.2. *Romulus.* Romulus [IKMP] is a finalist in the NIST lightweight cryptography standardization project. It is based on the SKINNY TBC [BJK+16] and incurs very

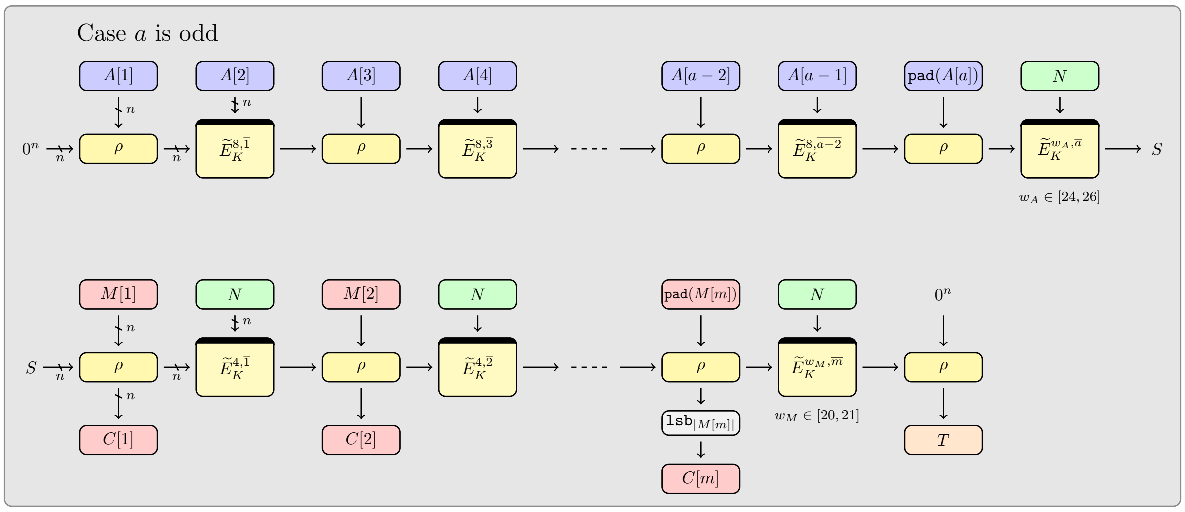
Case a is odd
$A[1]$
$A[2]$
$A[3]$
$A[4]$
$A[a-2]$
$A[a-1]$
pad($A[a]$)
N
0^n
ρ
$\widetilde{E}_K^{8,\bar{1}}$
$\widetilde{E}_K^{8,\bar{3}}$
$\widetilde{E}_K^{8,\overline{a-2}}$
$\widetilde{E}_K^{w_A,\bar{a}}$
S
$w_A \in [24, 26]$
$M[1]$
$M[2]$
pad($M[m]$)
$\widetilde{E}_K^{4,\bar{1}}$
$\widetilde{E}_K^{4,\bar{2}}$
$\widetilde{E}_K^{w_M,\bar{m}}$
$w_M \in [20, 21]$
lsb$_{|M[m]|}$
$C[1]$
$C[2]$
$C[m]$
T
n

FIGURE 1. The Romulus-N AEAD scheme [rom].

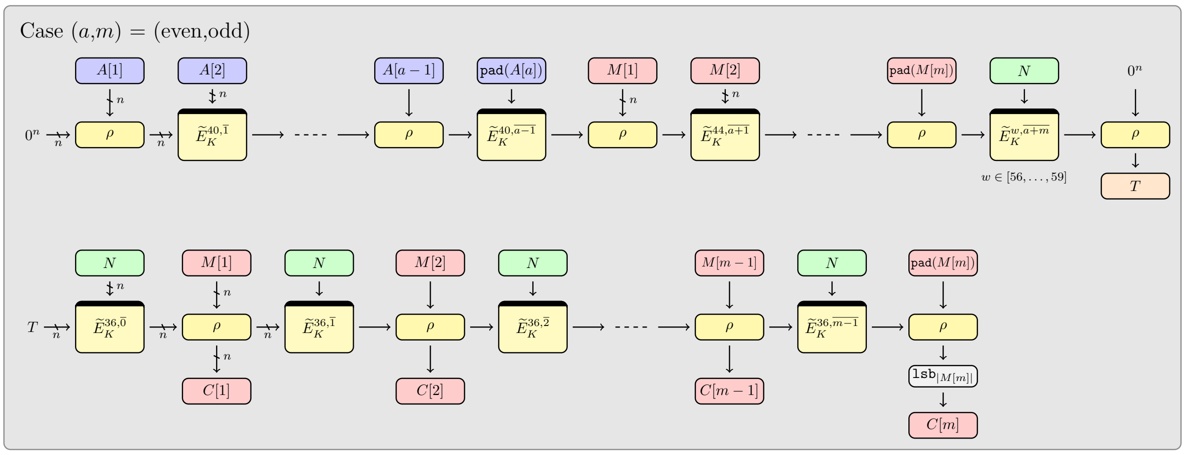
Case (a,m) = (even,odd)
$A[1]$
$A[2]$
$A[a-1]$
pad($A[a]$)
$M[1]$
$M[2]$
pad($M[m]$)
N
0^n
$\widetilde{E}_K^{40,\bar{1}}$
$\widetilde{E}_K^{40,\overline{a-1}}$
$\widetilde{E}_K^{44,\overline{a+1}}$
$\widetilde{E}_K^{w,\overline{a+m}}$
$w \in [56, \ldots, 59]$
T
$\widetilde{E}_K^{36,\bar{0}}$
$\widetilde{E}_K^{36,\bar{1}}$
$\widetilde{E}_K^{36,\bar{2}}$
$M[m-1]$
$\widetilde{E}_K^{36,\overline{m-1}}$
$C[1]$
$C[2]$
$C[m-1]$
lsb$_{|M[m]|}$
$C[m]$
ρ
n

FIGURE 2. The Romulus-M AEAD scheme [rom].

small overhead in terms of storage compared to the underlying TBC. The schemes presented, however, are independent of the underlying TBC [IKMP20]. In this paper, we focus on two variants of the Romulus family:

(1) *Romulus-N* is a nonce-respecting AEAD scheme, depicted in Figure 1.
(2) *Romulus-M* is a misuse-resistant AEAD scheme, depicted in Figure 2. Additionally, it offers security even if unauthenticated plaintext is released during decryption, which is a security model targeted for hardware accelerators with small memory [ABL$^+$14].

Romulus-N is the main variant of the Romulus family, and almost all of the analysis and benchmarking have been done on this variant. However, Romulus-N fails under the security threats of nonce-reuse and Release of Unverified Plaintext (RUP). Romulus-M, on the other hand, ensures security even in the presence of nonce-misuse and RUP.

2.1.3. *Tweakable Block Ciphers.* A TBC is a keyed function $\tilde{E}$ that takes three inputs: a secret key K, a public tweak T and an n-bit message block M. For each selection of (K, T), it behaves as a random permutation over the space of n-bit binary strings. A TBC is superior to a classical BC due to the extra public tweak T. TBCs have been formalized in [LRW02] and have been a useful tool in the field of SKE encryption and design for years. Since the introduction of the ΘCB-3 [KR11] and the design of the Deoxys-BC [JNPS16], more and more research have been performed in order to design practical, efficient and cheap TBCs and TBC-based schemes. Recently, two TBCs have been considered for the final stages of ISO standardization; Deoxys-BC and SKINNY. Deoxys-BC is the basis of Deoxys, the winner of the CAESAR competition for AEAD designs. SKINNY, on the other hand, is the basis for Romulus, a finalist in the NIST lightweight cryptography standardization project.

2.1.4. *Implementations.* The implementations explored in this paper are built using the Verilog HDL language, and studied using the iVerilog [Wil] and VCS simulators and the Synopsys Design Compiler synthesizer. The technology targeted is TSMC 65nm. The implementations will be made available as open source with the final version of this paper.

2.2. **ISO/IEC 18033-7: Deoxys-BC and SKINNY.** The ISO/IEC 18033-7 is currently in the *"under publication"* stage. It specifies the two TBC families: SKINNY and Deoxys-BC. In this section, we look at there combinational cost of there largest variants, with block size of 128 and tweakey size of 384. We assume they are used either as CPU co-processor or inside an iterative hardware accelerator that operates on the full 512-bit state in a single clock cycle. Hence, the smallest implementation is the round-based implementation; the combinational circuit performs 1 round per call/clock cycle. Deoxys-BC and SKINNY share similar design principles and they are both based on the tweakey framework. They are both based on the STK framework [JNP14] and Substitution-Permutation Networks (SPNs). Each round consists of 4 operations: SBoxLayer, AddRoundTweakey, ShiftRows, and MixColumn, with slightly different order. Deoxys-BC has an extra AddRoundTweakey operation in the final round. In this paper, we focus on the largest variant of each family. SKINNY-128-384+ consists of 40 rounds, while Deoxys-BC-128-384 consists of 16 rounds. However, since the internal components are different, SKINNY's round function is a lot smaller than that of Deoxys-BC. A common misconception is a cipher with less rounds is faster than a cipher with more rounds. This is not true, as the definition of a round is arbitrary and our experiments show that SKINNY can be faster than Deoxys-BC in many use-cases. While SKINNY is targeted towards lightweight and low-area applications, our experiments show that Deoxys-BC is only faster than SKINNY at very high frequencies, and this comes at a huge area cost. Besides, such high frequencies maybe unachievable. The cipher will be used inside a processor or a hardware accelerator, and the frequency will be decided by other parts of the system; *e.g.* CPU, Finite State Machine (FSM)...*etc.*

We assume that all calls have the same number of rounds, such the combinational circuit of SKINNY-128-384+ can perform 1, 2, 4, 5, 8, 10, 20 or 40 rounds, while the circuit of Deoxys-BC-128-384 can perform 1, 2, 4, 8, or 16 rounds. Since Deoxys-BC uses the AES round function, we considered two possible configurations for the SBox: Look-Up Table (LUT), which is more suitable for high-speed and FPGA implementations, as discussed in [KCP17], and a low-area SBox circuit [boy, Hus], which suitable for low-area ASIC implementations. Table 1 shows the synthesis results for these configurations on the TSMC 65nm standard cell library. While the speed of a hardware accelerator or an SoC is not determined just by the cryptographic combinational circuit, this section can be viewed as investigating the limits of these TBCs. For example, Table 1 shows that the circuit cannot be computed on the considered technology in with less than 17.4ns, while Deoxys-BC-128-384 cannot computed in less than 12.91ns. When we add a 0.5ns safety slack per call, these numbers grow to 19.97ns and 13.4, respectively. However, when we also compare the area, we see that the faster implementation of Deoxys-BC-128-384 comes at a huge area cost, where the fastest implementation of SKINNY-128-384+ costs 2228.75 and 52892.5, respectively, while for Deoxys-BC-128-284, the best latency is achieved at 174169.24 GE. SKINNY-128-384+ also has the potential to be 7 times more efficient than Deoxys-BC-128-384 at high frequencies, and 2.3 times more efficient at 24 MHz. At low frequencies, the efficiency of both TBCs becomes almost constant, in which case they both offer an interesting straight-forward trade-off between speed and area. Note that these values are only for the combinational part of the cipher, and the efficiencies will drop when the storage is added. However, we exclude the storage from this section as the storage should be part of the higher-level system. In the next section, we will revisit how SKINNY-128-384+ and Deoxys-BC-128-384 can be used inside existing systems without requiring any extra storage.

Table 1. Comparison of the logic circuit of Skinny and Deoxys-BC-128-384 for different value of latency. Synthesis results are using TSMC 65nm.

TBC	**# of Rnds**	**# of Cycles**	**Area (GE)**	**Critical Path (ns)**	**Min. Latency (ns)**	**Safe Latency (ns)**	$\frac{128000}{L\times A}$ **at min. latency.**	$\frac{128000}{L\times A}$ **at 24MHz**
	1	40	1107	0.44	17.6	37.6	6.57	0.07
	2	20	2228.75	0.87	17.4	27.4	3.3	0.07
	4	10	4790.5	1.94	19.4	24.4	1.37	0.06
Skinny	5	8	6029.5	2.56	20.48	24.48	1.03	0.06
	8	5	10117	3.86	19.3	21.8	0.66	0.06
	10	4	12776.75	4.94	19.76	21.76	0.51	0.06
	20	2	26554.68	9.78	19.56	20.56	0.25	0.06
	40	1	52892.5	19.47	19.47	19.97	0.12	0.06
	1	16	9825.25	0.88	14.08	22.08	0.93	0.02
Deoxys-BC-128-384	2	8	20998	1.7	13.6	17.6	0.45	0.02
	4	4	43588.5	3.4	13.6	15.6	0.22	0.02
LUT SBox	8	2	86156.25	6.65	13.3	14.3	0.11	0.02
	16	1	174169.24	12.91	12.91	13.4	0.06	0.02
	1	16	7347.25	1.26	20.16	28.16	0.86	0.03
Deoxys-BC-128-384	2	8	19346	2.39	19.12	23.12	0.35	0.02
	4	4	43634	4.43	17.72	19.72	0.17	0.02
Low Area SBox	8	2	91086.11	8.78	17.56	18.56	0.08	0.02
	16	1	181776	16.71	16.71	17.21	0.04	0.02

2.3. **Romulus-N/M Hardware Accelerator.** The proposed architecture compliant with the lightweight cryptography hardware API proposed in [KDT+19]. This choice is to allow fair comparison to other implementations and to allow a full implementation that does not ignore any hidden costs such as key storage, nonce storage or message padding. The architecture is parameterized by the following parameters:

- w: bus width, defaulted to 32 bits.
- S_s: number of state shares; used for masked implementations, defaulted to 1.
- K_s: number of secret key shares; used for masked implementations, defaulted to 1.
- TBC: choice of the TBC.
- r: number of TBC rounds per cycle.

The architecture, depicted in 3, is built based on 4 register files:

1. State Register File (SRF): consists of $128 \times S_s/w$ words, each consists of w bits.
2. Key Register File (KRF): consists of $128 \times K_s/w$ words, each consists of w bits.
3. Tweak Register File (TRF): consists of $128/w$ words, each consists of w bits. Since the tweak is always public, it does not need to be masked.
4. Counter Register File (CRF): consists of $128/w$ bits.

The SRF is reset to 0 at the beginning a new encryption or decryption instruction. It is loaded in the feedback mode, where the G function is applied to the bottom word in a byte-wise fashion. It transforms each byte as follows:

$$(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7) \rightarrow$$

$$(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_0 \oplus x_7)$$

The output of this transformation is XORed with the input word (from the control unit) to generate the plaintext during encryption and the ciphertext during decryption. The plaintext (the input word during authentication and encryption, and the output of the previous XOR during decryption) is XORed with bottom word to generate the feedback word that get fed from the top, shift all the other words down. In order to support Romulus-M, a bypass connection is needed such that the tag from the top operation in Figure 2 is fed back into the SRF. This bypass connection is shown in red in Figure 3. The KRF and TRF are similar, where they are loaded from the top down, with no feedback needed. The CRF is used to hold the block counter and the domain separator; the constant part of the public tweak. It does not need a load operation, as it is reset to the initial value of the counter. Each register file can also be read and written in parallel, where the circuit from Section 2.2 is represented in Figure 3 by the combination of TBC, Key Schedule, Tweak Schedule and Domain Separator Schedule.

During execution, the nonce will be stored in the TRF, while the secret key will be stored in the KRF. There values will be changed, and needs to be corrected during, which the purpose of the Secret Key Correction and Tweak Correction circuits. These operations, alongside the Counter, are performed in between the TBC calls, in parallel to loading the SRF.

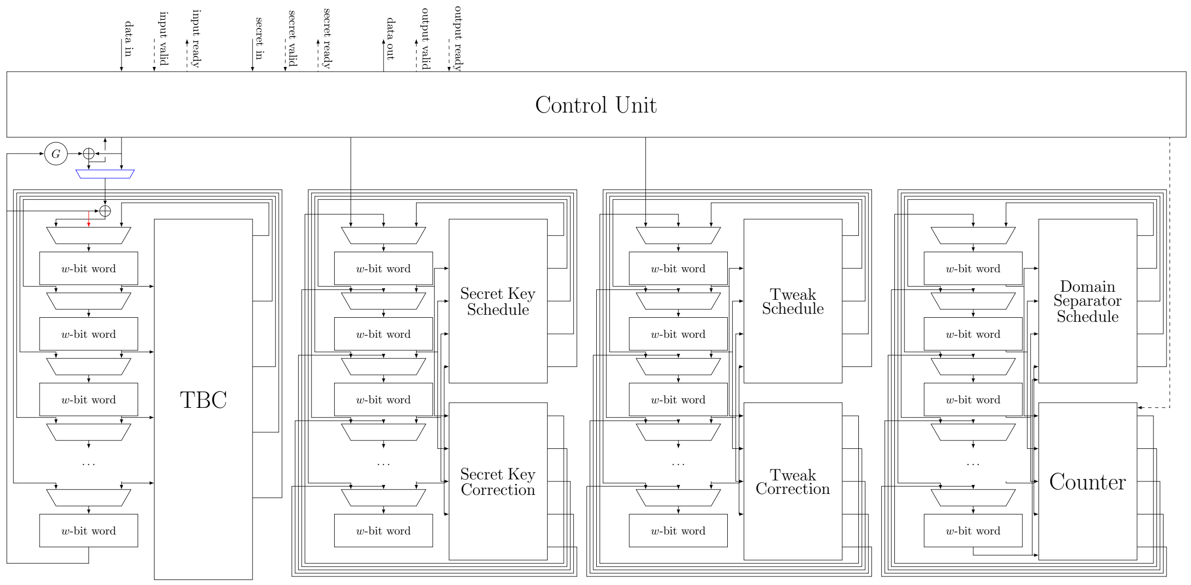

FIGURE 3. The architecture of the proposed hardware accelerator. Solid arrows are w-bit wide, where w is the configurable bus width. The dotted arrows are control signals. Some of the control signals are omitted from the diagram for simplicity including multiplexers' selectors, registers enables and resets. The red arrow is only needed for Romulus-M and can be removed if Romulus-M support is not required. The blue multiplexer is used to switch between encryption and decryption.

Table 2 shows the latencies of the proposed architecture for both Romulus-N and Romulus-M encryption instructions, for different sizes on A and M, where a and m are the number of bytes of A and M, respectively. This is shown when $w = 32$, and both the state and key are unmasked. The latencies do not depend on the TBC used. The bottom section of Table 2 includes the ratio between the latency of Romulus-M and Romulus-N, where for short messages (64 bytes of A and M) and low area (high latency), the ratio is about 1.3, while for longer message low area configurations the ration is less than 1.4. For empty A, the ratio is less than 1.65 for low area configurations.

Table 2. Latency of Romulus-N and Romulus-M for different latencies of the TBC. $w = 32$

Mode	a	m	40	20	10	4	1
N	0	0	101	61	41	29	23
	0	64	237	137	87	57	42
	64	0	197	117	77	53	41
	64	64	333	193	123	81	60
	0	1152	3225	1765	1035	597	378
	1152	0	1825	1065	685	457	343
	1152	1152	4949	2769	1679	1025	698
M	0	0	101	61	41	29	23
	0	64	338	198	128	86	65
	64	0	157	97	67	49	40
	64	64	434	254	164	110	83
	0	1152	4954	2774	1684	1030	703
	1152	0	1785	1045	675	453	342
	1152	1152	6678	3778	2328	1458	1023
Latency Overhead	0	0	1.00	1.00	1.00	1.00	1.00
	0	64	1.43	1.45	1.47	1.51	1.55
	64	0	0.79	0.83	0.87	0.92	0.98
	64	64	1.30	1.32	1.33	1.36	1.38
	0	1152	1.57	1.57	1.63	1.72	1.86
	1152	0	0.99	0.98	0.98	0.99	1.00
	1152	1152	1.35	1.36	1.39	1.42	1.47

Table 3 shows the synthesis results for a group of unmasked configurations for both SKINNY-128-384+ and Deoxys-BC-128-384, for both maximum frequency and low area 24 MHz constraints. Our experiment shows that with SKINNY, both Romulus-N and Romulus-M achieve the minimum energy consumption when the TBC latency is 10 cycles (4 rounds per cycle), while with Deoxys-BC-128-384 the minimum energy is achieved when the TBC latency is 16 (one round per cycle). Low latency implementation (1 clock cycle per TBC) can be achieved using SKINNY-128-384+ (40 rounds) in around 55 kGE. Besides, for around the same area ($\approx$ 12 kGE), Romulus with SKINNY-128-384+ is 1.85 and 1.7 times fast than Romulus with Deoxys-BC-128-384 for N and M, respectively. For high frequencies, Romulus with Deoxys-BC-128-384 is 1.24 and 1.34 faster for N and M, respectively. At 24MHz, the single-cycle implementation of SKINNY-128-384+ can achieve relatively high speeds.

2.4. **Comparison with Lightweight AEAD Accelerators.** The authors of [KPC20] have compared some the NIST lightweight cryptography candidates [TMC+21]. We reproduced their results for most of the NIST lightweight cryptography candidates as well

TABLE 3. Synthesis result of the proposed hardware accelerator on TSMC 65nm.

		24MHz						High Speed					
TBC Rounds	**Area (GE)**	**CP (ns)**	**Power (mW)**	**Th/put (Mbps) N**	**Th/put (Mbps) M**	**Energy (pJ/bit) N**	**Energy (pJ/bit) M**	**CP (ns)**	**Power (mW)**	**Th/put (Gbps) N**	**Th/put (Gbps) M**	**Energy (pJ/bit) N**	**Energy (pJ/bit) M**
SKINNY-128-384+													
1	7348.61	1.11	0.22	89	66	2.47	3.33	1.5	0.74	2.48	1.84	0.30	0.40
2	7865.28	1.16	0.24	159	117	1.49	2.04	1.5	0.74	2.43	3.25	0.17	0.23
4	10124.24	1.9	0.32	263	190	1.21	1.67	2.0	0.70	5.49	3.96	0.13	0.18
5	12035.49	2.41	0.38	302	217	1.26	1.78	2.5	0.73	4.06	2.87	0.13	0.18
10	17767.99	5.03	0.59	431	303	1.36	1.94	5.2	0.73	3.46	2.43	0.21	0.30
40	55098.25	19.25	1.94	633	432	3.06	4.47	20	1.96	1.32	2.43	1.48	2.17
Deoxys-BC-128-384													
1	12659.75	1.91	0.431	163	127	2.47	3.33	1	0.74	6.81	5.31	0.30	0.40

as the implementation of AES-GCM from [gmu]. All the implementations are complying with the same interface, except AES-GCM, which adopts an earlier closely related interface. Our goal is to minimize the cost while having competitive performance. In order to do so, we focus the comparison on the Energy×Area product. Figures 4 and 5 show the comparison between, Romulus-N-SKINNY, Romulus-M-SKINNY, Romulus-N-Deoxys-BC, Romulus-M-Deoxys-BC, AES-GCM and other NIST lightweight cryptography finalists. The considered configurations are one round per cycle for Deoxys and the minimum Energy×Area product for SKINNY-128-384+. The results show that Romulus-M is very close to with Romulus-N. When using SKINNY, only TinyJambu outperforms Romulus. Besides, even when using Deoxys-BC-128-384, the performance is still comparable to the NIST lightweight candidates. Based on these observation and the fact that Romulus-M is the only misuse-resist scheme for both privacy and integrity, it is recommended that sensitive applications consider Romulus-M-SKINNY or Romulus-M-Deoxys-BC as a viable and secure option.

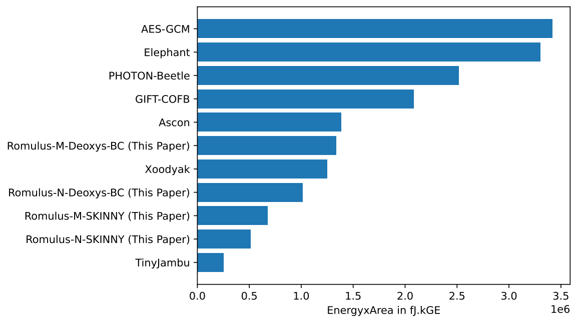

FIGURE 4. Energy×Area product for 64 bytes of A and M for various AEAD schemes.

3. FPGA EXPERIMENTS ON FIRST-ORDER MASKING OF THE SKINNY 8-BIT SBOX

Masking is one of the oldest, and yet most relevant, countermeasures against statistical Side-Channel Analysis (SCA) of cryptographic implementations. Several efforts over the years have been performed to improve the security and efficiency of masked ciphers. On the other hand, the security models of masked ciphers has been improving to better capture the side-channel leakage of real-world implementations. In particular, work on masked ciphers can be classified into 4 categories:

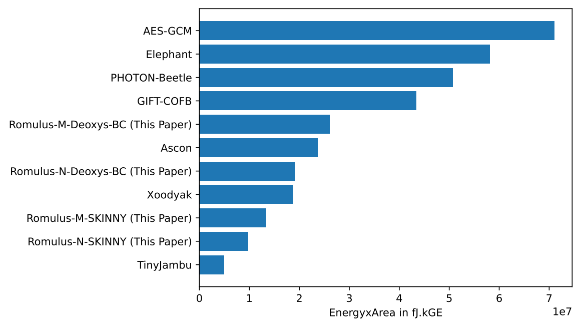

FIGURE 5. Energy×Area product for 1536 bytes of A and M blocks for various AEAD schemes.

(1) Design and implementation of secure masking schemes, *e.g.* ISW [ISW03], Threshold Implementations (TI) [BGN+14], Domain-Oriented Masking (DOM) [GMK16], Hardware Private Circuits (HPC) [CGLS20].
(2) Modeling hardware and software implementations in order to have better understanding of their behavior, *e.g.* the probing model, Non-Interference (NI), Strong Non-Interference (SNI) [BBD+16], Probe-Isolating Non-Interference (PINI) [CS20].
(3) Attacking masked implementations using SCA techniques, *e.g.* Differential Power Analysis (DPA) [KJJ99], Correlation Power Analysis (CPA) [BCO04].
(4) Designing evaluation frameworks in order to measure the security order of different circuits and implementations, *e.g.* `maskVerif` [BBC+19] and `SILVER` [KSM20] for formal verification of circuits and frameworks based on statistical test (TVLA [SM15], χ^2-test [MRSS18]) for assessing practical leakages.

However, one area that remains roughly under-studied is the hardware implementations of masked ciphers. Since the introduction of masking as a countermeasure, researchers have shown shortcomings of the idealized security model. Initially, masking schemes were designed to be secure in the probing model, i.e., a d^{th}-order masked implementation should be secure as long as the adversary can observe at most d internal wires of the circuit. It was shown that hardware glitches can lead to problems in this model. By hardware glitches, we refer to imbalances in the physical delay of different paths of the same circuit. Such imbalances can lead to exposing more than d variables by observing only $\leq d$ wires. In order to overcome such issues, new security models were introduced, including glitch-extended probing model, NI, SNI and PINI.

Glitches are not the only physical anomaly that leads to a gap between theoretical models and physical implementations. Two problems have been observed in several works are transitions and coupling. Transitions refer to shared variables being recombined due to the transition of values inside flip-flops, while coupling refers to dependencies in the electric current of two or more wires that are close to each other and that should carry independent variables. Coupling in particular can be very problematic. The probing security model is built on a fundamental assumption that the leakage observed from different independent wires of the circuit are independent. However, it has been observed time and time again that a correct masked implementation (of any order) experience first-order leakage [DCEM18] when implemented on FPGA. The authors of [DCEM18] have analyzed this issue and provided explanations from a VLSI point of view. Besides, the authors of [LBS19] have shown that this issue can lead to attacks on masked implementations. While ASIC implementations are less studied from this point of view, it is more conservative to assume the same issue will arise in an ASIC implementation.

In order to address this issue and provide meaningful benchmarking, ASIC and FPGA implementation design should be performed hand in hand, with FPGA practical evaluation guiding ASIC implementations. Of course, with access to a real-world practical ASIC chip a more tailored approach can be performed. However, for benchmarking purposes, it is better to take the more conservative approach and apply it to different designs in the same manner.

Another issue facing benchmarking of SCA-protected implementation is defining and assigning a security order for a given implementation. The widely accepted definition of first-order security is using Welch's *fixed vs. random* t-test. The test in its basic form compares two populations of samples; one with a fixed plaintext and the other with randomly sampled plaintexts, against an equality of mean assumption. The test statistic is given by

$$t = |\frac{\mu_f - \mu_r}{\sqrt{\frac{s_f^2}{N_f} + \frac{s_r^2}{N_r}}}|$$

A widely accepted threshold for the t-value is 4.5, where samples that lead to higher values are said to exhibit observable first-order leakage. In other words, first-order leakage is observable when the mean of observed samples is data-dependent. When it comes to higher-order leakage, the definition becomes more ambiguos. Two versions of the t-test are used to detect higher-order leakage. The univariate higher-order test is performed in two steps. First, preprocessing the samples, where for each sample $L_x(i)$ in the population $x \in \{f, r\}$,

$$L_x^{'}(i) = (L_x(i) - \mu_x)^o$$

where o is the order of the test. Second, the first-order t-test is performed on the preprocessed samples.

$$t = |\frac{\mu_f^{'} - \mu_r^{'}}{\sqrt{\frac{(s^{'})_f^2}{N_f} + \frac{(s^{'})_r^2}{N_r}}}|$$

The other higher-order t-test is the multivariate t-test. The first step, is performed as follows

$$L_x^{'}(N_x * i + j) = (L_x(i) - \mu_x) \times (L_x(j) - \mu_x) \forall 0 \leq i, j \leq (N_x - 1)$$

The two versions of the t-test differ in what they measure and also their computational cost. The univariate test captures leakage in higher statistical moments of the samples, while the multivariate test captures leakage that requires combining samples from different timestamps. When it comes to computational complexity, in the worst case it include calculating the the mean of each sample, exponentiation, and the first order t-test. If the cost of exponentiation is c_e, the length of each trace is s, and the cost of the first order test is c_1, then the upper bound of the cost of the univariate higher-order test is

$$c_o \leq c_1 + s \cdot (N_f + N_r) \cdot c_e + s \cdot (c_{\mu_f} + c_{\mu_r}) \leq 2 \cdot c_1 + s \cdot (N_f + N_r) \cdot c_e$$

For small order, o and c_e are small. Hence, the univariate higher-order test is not significantly more complex than the first order variant, *i.e.* $c_o \leq a \cdot c_1$, where a is a small constant. On the other hand, the multivariate test increases the complexity significantly, since the number of samples per trace increases from s to s^o. This exponential growth makes the test very expensive for higher orders.

Besides, the χ^2-test has been proposed [MRSS18] to detect higher order univariate leakage, *i.e.* leakage not based on the difference of means. The cost of χ^2-test is comparable to that of the univariate t-test. Hence, implementations that pass higher-order tests do not offer a significant security gains compared to implementations that pass the first-order t-test but fail higher-order tests. The difference between univariate and multivariate t-test is captured by whether the combined information come from leakage at the same point in time or not. In other words, univariate higher-order leakage can usually be observed in implementations that processes the different shares of the masked implementation in parallel.

For a standardization project such the NIST lightweight cryptography project, benchmarking has to be done on a variety of platforms. The main platforms for hardware benchmarking are FPGAs and ASIC. Since the manufacturing cost of ASICs is too high and sometimes prohibitive for academic projects, ASIC benchmarking is usually done at a pre-fabrication level. However, when it comes to SCA, we need to question whether this high-level abstraction is sufficient. On the other hand, FPGAs are more accessible in a lab settings and FPGA boards for SCA evaluation are available in most labs. While FPGAs do not represent most of the use cases of lightweight cryptography, they can be used as a starting point for evaluation.

Contributions

In this work, we look at the design of first-order hardware masked implementations. We perform a two-part practical study on the design of SBoxes using established masking schemes. In the first part, we implement the SKINNY 8-bit SBox using DOM, DOM-SNI, HPC, CMS, PARA, PINI and TI. We analyze these implementations using the formal leakage assessment tool SILVER. We show the different security assumptions they fulfill. In the second part, we study the SBoxes in practice using the Sasebo-GII FPGA board, electromagnetic leakage and TVLA. We show that all these SBoxes are vulnerable to first-order leakage with very small number of traces.

We also develop a framework for using TVLA for unit testing. Unit test is an integral part of digital hardware design. However, it is not always taken care of in the early stages of design due to the practical challenges and complexity. Our framework can measure and process 3 million traces in 1 hour, allowing evaluating 200 million traces in less than 3 days. Since weak implementations are expected to leak data after a few million traces, this speed-up allows testing many different ideas and implementations in short amount of time.

Finally, given the weakness of the implementations considered, we propose a new method of implementing DOM, targetting SNI first-order security in practice. The proposed method suffers from a $4\times$ slow-down. However, it offers at least 3 orders of magnitude better protection against straightforward implementations.

3.1. **Designing secure implementations using SILVER.** The SKINNY 8-bit SBox consists of 8 NOR gates and 8 XOR gates in an iterative Feistel structure, as depicted in Figure 6. In particular, it can be viewed as four iterative rounds where each round consists of 2 NOR gates and 2 XOR gates followed by a bit permutation. A naive implementation would simply pipeline this structure with 8 Flip-Flops after each round. Note that each round has only two parallel non-linear gates, so one pipeline stage per round should be sufficient to thwart side-channel leakage based on glitches. However, for lightweight applications, this implementation could be expensive as it requires *at least* $32 \times d$ flip-flops, irrespective of the masking scheme, where d is the number of

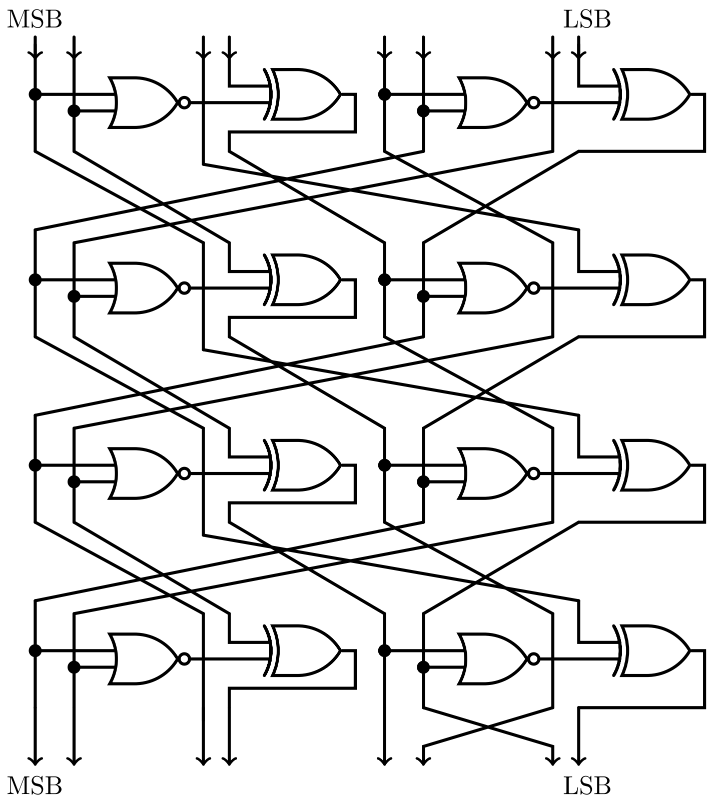

FIGURE 6. The SKINNY 8-bit SBox.

shares. For serial modes, pipelining of the SBox is not required. Hence, we change the representation of the SBox to suit sequential evaluation of the SBox without full pipelining. We note that the SBox operation is basically the sequential evaluation of one core function:

$$a \leftarrow \neg(x \vee y) \oplus z$$

We can then untangle to the SBox into the following iterative process:

$$a_0 \leftarrow \neg(b_7 \vee b_6) \oplus b_4$$
$$a_1 \leftarrow \neg(b_3 \vee b_2) \oplus b_0$$
$$a_2 \leftarrow \neg(b_2 \vee b_1) \oplus b_6$$
$$a_3 \leftarrow \neg(a_0 \vee a_1) \oplus b_5$$
$$a_4 \leftarrow \neg(a_1 \vee b_3) \oplus b_1$$
$$a_5 \leftarrow \neg(a_2 \vee a_3) \oplus b_7$$
$$a_6 \leftarrow \neg(a_3 \vee a_0) \oplus b_3$$
$$a_7 \leftarrow \neg(a_4 \vee a_5)xx \oplus b_2$$

where $b_7 ... b_0$ are the 8 input bits, and the 8 output bits are $s_7 s_6 \cdots s_0$, and assigned using the following permutation

$$s_6 s_5 s_2 s_7 s_3 s_1 s_4 s_0 \leftarrow a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7$$

This modularity makes the task of masking the SBox easier in two regards. First, we can focus on masking only the core function. Second, we can check the dependency between the outputs of the core function and determine the way to implement with the minimum number of flip-flops. We note that a_0, a_1 and a_2 depend only on input bits, so the can be computed in the first iteration in parallel. a_3 and a_4 depend on the previous 3 bits and are independent of each other, so the can be computed in the second iteration. Similarly, a_5 and a_6 are computed in the third iteration, while a_7 is computed in the fourth iteration.

Masking the core function depends on the masking scheme used. However, since most masking schemes are designed with AND and XOR gates in mind, a useful transformation to the core function is to change its representation to

$$a \leftarrow (\neg x \wedge \neg y) \oplus z$$

which computes exactly the same value, but helps visualize the impact of using different masking schemes. Depending on the masking scheme used to mask the core function, one iteration can consist of one or more cycles, leading to SBoxes that require 4 or more cycles, respectively.

Formal verification

In order to verify our security goals for each of the considered implementations, we have performed formal verification of the gadgets used and the full SBox implementations using the Statistical Independence and Leakage Verification (SILVER) tool proposed Knichel, Sasdrich and Moradi [KSM20]. The tool tests the formal properties of gate-level netlists of different circuits. The netlist is generated using the Yosis open source synthesis tool. The tool tests the circuits against 4 security models: probing, NI, SNI and PINI, with different security orders. Table 4 shows the results obtained for each of the masked implementations tested for each security model, where + means secure with glitches, y means secure without glitches and - means not secure.

Table 4. Formal verification results using SILVER

Implementation	**Random Bits**	**Cycles**	**Probing**	**NI**	**SNI**	**PINI**
DOM	8	4	y	y	y	-
DOM Full FFs	8	4	+	y	y	-
DOM Dep.	16	4	?	?	?	?
DOM SNI	8	8	+	+	+	-
DOM Rapid	25	2	?	?	?	?
CMS	32	4	+	+	y	-
CMS Rapid	76	2	?	?	?	?
ISW	8	8	+	+	+	-
ISW PINI	16	12	+	+	+	+
HPC	8	8	+	+	+	y
HPC STR	16	12	+	+	y	+
PARA	16	8	+	+	+	-
PINI	8	4	y	y	y	y

3.2. **Practical Testing. Testing Set-Up** We have designed a unit testing framework for high speed testing of SBoxes at the early phases of designing RTL code. The framework is shown in Figure 7. A Deterministic Random Bit generator (DRNG) based on a low-latency implementation of a block cipher in the counter mode is used to generate the masked shares and decide whether the next sample is fixed or random. A gardening period of 200 cycles is applied between generating the shares and updating the SBox, to avoid leakage from DRNG circuit. The Oscilloscope sends a START command, which triggers the setup to calculate 15,000 samples. After 15,000 samples, the set-up halts, allowing the oscilloscope to synchronize and store the acquired traces. The samples are calculated in less than a second, and storing them requires about 18 seconds. Hence, it takes about 1 hour to calculates and process 3 million traces.

On top of that, in order to reduce the trace acquisition complexity, we need to artificially raise the Singal-to-Noise Ratio (SNR). We propose to do so by replicating the Unit-Under-Test (UUT) multiple times, forcing the FPGA to perform the same logic multiple times simultaneously. This can have an amplifying effect similar to computing the same trace multiple times and averaging out the noise, which was proposed in [Sta18]. However, it achieves the same effect with a lot less traces. This enables very fast early testing of SBoxes, where with $9\times$ replications, most SBoxes can be broken with a few thousand traces.

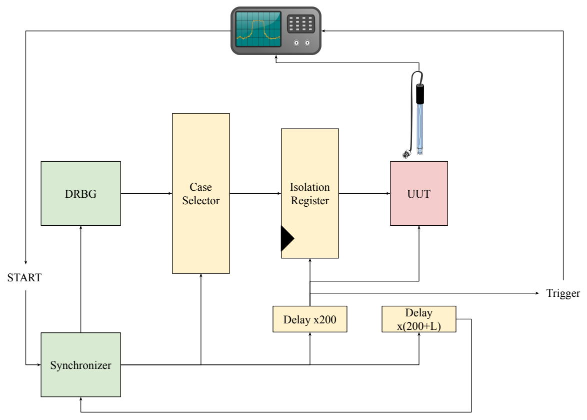

FIGURE 7. The proposed framework/set-up for unit leakage assesment

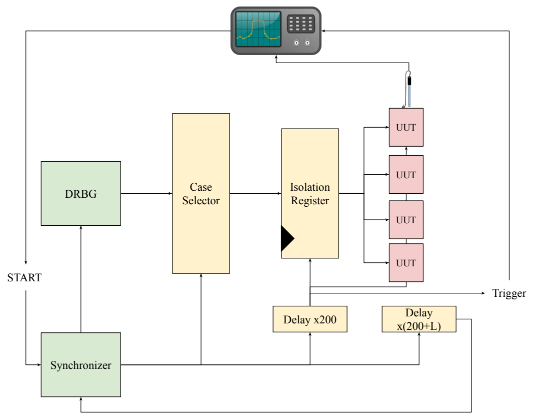

FIGURE 8. The measuring the leakage of replicated UUTs.

3.3. **The Practical Insecurity of Hardware Masking.** A common assumption is that the leakage observed by the adversary/evaluator is the sum of independent leakage components. Hence, an adversary observing the combined leakage

$$\mathsf{L} = \mathsf{L}_0 + \mathsf{L}_1 + \cdots + \mathsf{L}_{d-1}$$

cannot detect any first-order leakage, as each of the components is independently distributed. However, if was shown De Cnudde *et al.* [DCEM18] that in FPGA this assumption may not be true, due to what became later known as coupling, where the power consumption and power supply noise from one share, *e.g.* L_0 may impact the power consumption of the other shares. Consequently, they observed first order leakage on FPGA for several circuits, including masked AES MixColumn and SBox. In [LBS19], it was shown that such coupling effects can be exploited in SCA attacks, and not just observable leakage. The authors of [DCEM18] proposed some suggestions of how to mitigate this issue, which include VLSI-based solutions to isolate the power supplies to different shares and serializing implementations in order to achieve *"hardware non-completeness"*, where at any point in time only a single share is being processed.

While VLSI-based solutions may mitigate the coupling issue for first-order leakage, we argue in this paper that for certifiable implementations, a.k.a implementations the claim a given security order, that hardware non-completeness should be considered a goal.

Serializing masked implementation can lead to transitional leakage. For example: A flip-flop holding the value x^0 may be updated at some point with x^1, which can lead to leakage that depends on both shares. The same thing can happen in logic gates, where unintended values can be processed by the logic gates, leading to unintended share combining.

Power Gating [JMSN05]

Power-gating is a technique used for low-power digital design, where a logic circuit that is not frequently used can be turned off by forcing all its inputs and outputs to 0. It is usually combined with clock gating, where the clock is turned off for flip-flops that are not being updated.

Non-Complete DOM Implementation

In order to implement DOM such that only one share of each variable is being processed at a time, we treat each sub-share during the internal computation as a tiny power domain, having its own enable/disable signal. The share computation operation is updated as shown below.

$$s^3 \leftarrow e_0 \wedge ((e_0 \wedge x^1) \wedge (e_0 \wedge y^1) \oplus (e_0 \wedge z^1)$$

$$s^2 \leftarrow e_1 \wedge ((e_1 \wedge x^1) \wedge (e_1 \wedge y^0) \oplus (e_1 \wedge r))$$

$$s^1 \leftarrow e_2 \wedge ((e_2 \wedge x^0) \wedge (e_2 \wedge y^1) \oplus (e_2 \wedge r))$$

$$s^0 \leftarrow e_3 \wedge ((e_3 \wedge x^0) \wedge (e_3 \wedge y^0) \oplus (e_3 \wedge z^0))$$

$$a^0 \leftarrow e_4 \wedge ((e_4 \wedge s^0) \oplus (e_4 \wedge s^1))$$

$$a^1 \leftarrow e_5 \wedge ((e_5 \wedge s^2) \oplus (e_5 \wedge s^3))$$

At any point in time, at most one of the signals e_0, e_1, e_2, e_3, e_4 and e_5 is set to 1. This helps reducing leakage in two ways:

(1) Making sure both shares of the same variable are never carried by two adjacent wires or gates and never stored into flip-flops at the same time, the coupling effect is removed to a very large extent.
(2) Reducing the power consumption of the SBox as a whole leads to less exploitable leakage.

In Table 5, we show the SNR and number of traces required for each method of implementing DOM. The results show that our proposed implementation is several orders of magnitude more secure than other methods. It requires about $1,000\times$ more traces for the same number of replications and $10\times$ higher SNR. With low SNR, the implementation did not show leakage even with more than 200 million traces. The t-value outcome for each of the implementations is show in Figures 9, 10, 11 and 12. We have also tested the 4-share 2-cycle first-order threshold implementation of the SBox proposed in [CCGB21] (TI33), showing that it is also vulnerable to this issue, and has not been tested enough. The code for TI33 is the same code used in the original paper.

Table 5. The masking schemes, number of UUT replicas, the SNR ($\frac{\mu}{\sigma}$) and number of traces required to fail the TVLA test.

Scheme	**Replicas**	**SNR**	**Traces**
Mask Off			
DOM-SNI	1	174.3	823
TI33	1	172.3	1,536
Mask On			
DOM	9	174.5	7,784
DOM-SNI	9	173.3	2,140
TI33	9	804.7	1,393
TI33	1	864.19	62,924
NC	99	2028.44	5,913,875
NC	9	1183.08	6,190,000
NC	1	33.57	>200,000,000

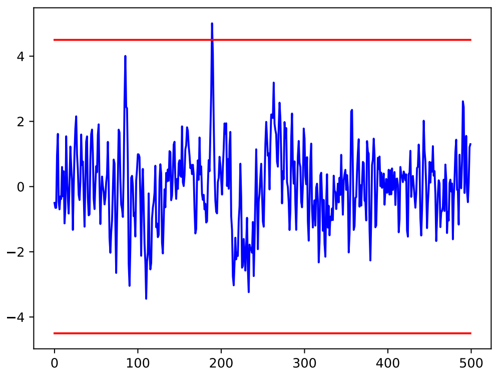

Figure 9. TVLA output for DOM with 9 replicas after 7,784 traces.

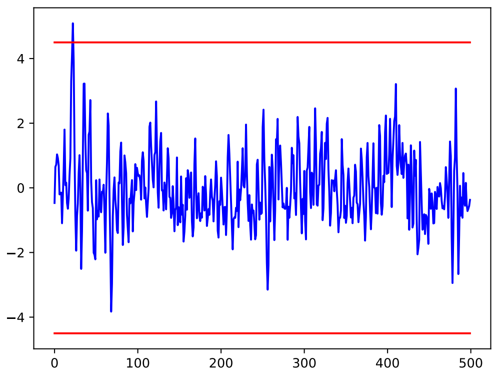

Figure 10. TVLA output for DOM-SNI with 9 replicas after 2,140 traces.

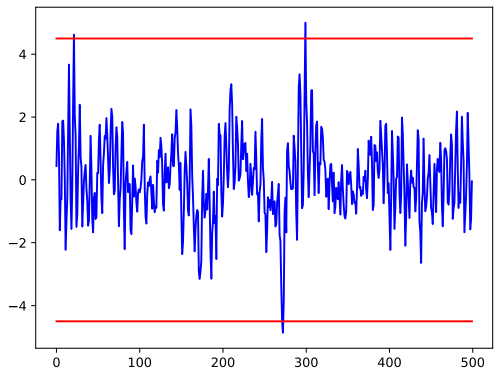

FIGURE 11. TVLA output for NC with 9 replicas after 6,190,00 traces.

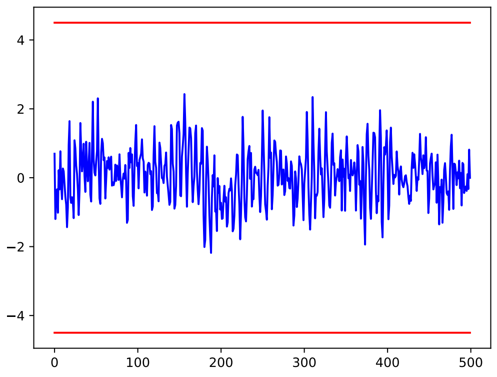

FIGURE 12. TVLA output for NC with only 1 replica after 210,000,000 traces.

3.4. **Future Work.** We are currently in the process of developing and testing the full Romulus-N/M accelerator with these masked SBoxes.

REFERENCES

[ABL+14] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to securely release unverified plaintext in authenticated encryption. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 105–125. Springer, 2014.

[BBC+19] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. maskverif: Automated verification of higher-order masking in presence of physical defaults. In *European Symposium on Research in Computer Security*, pages 300–318. Springer, 2019.

[BBD+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 116–129, 2016.

[BCDM21] Tim Beyne, Yu Long Chen, Christoph Dobraunig, and Bart Mennink. Elephant v2. NIST Lightweight Cryptography Project (2021). 2021.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *International workshop on cryptographic hardware and embedded systems*, pages 16–29. Springer, 2004.

[BGN+14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. A more efficient aes threshold implementation. In *International Conference on Cryptology in Africa*, pages 267–284. Springer, 2014.

[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 *Annual International Cryptology Conference*, pages 123–153. Springer, 2016.

[boy] Circuit Minimzation Work. http://cs-www.cs.yale.edu/homes/peralta/CircuitStuff/CMT.html.

[CCGB21] Andrea Caforio, Daniel Collins, Ognjen Glamočanin, and Subhadeep Banik. Improving first-order threshold implementations of skinny. In *International Conference on Cryptology in India*, pages 246–267. Springer, 2021.

[CGLS20] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. *IEEE Transactions on Computers*, 70(10):1677–1690, 2020.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Transactions on Information Forensics and Security*, 15:2542–2555, 2020.

[DCEM18] Thomas De Cnudde, Maik Ender, and Amir Moradi. Hardware masking, revisited. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 123–148, 2018.

[DR01] Joan Daemen and Vincent Rijmen. Reijndael: The Advanced Encryption Standard. *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, 26(3):137–139, 2001.

[GLL17] Shay Gueron, Adam Langley, and Yehuda Lindell. AES-GCM-SIV: specification and analysis. *Cryptology ePrint Archive*, 2017.

[GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. *Cryptology ePrint Archive*, 2016.

[gmu] Hardware Benchmarking of CAESAR Candidates. https://cryptography.gmu.edu/athena/index.php?id=CAESAR.

[Hus] Siam Umar Hussain. Tiny AES. https://github.com/siamumar/tinyAES.

[IKMP] Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. Romulus v1.3. NIST Lightweight Cryptography Project (2021).

[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*, pages 43–120, 2020.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *Annual International Cryptology Conference*, pages 463–481. Springer, 2003.

[JMSN05] Hailin Jiang, Malgorzata Marek-Sadowska, and Sani R Nassif. Benefits and costs of power-gating technique. In *2005 International conference on computer design*, pages 559–566. IEEE, 2005.

[JNP14] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Tweaks and keys for block ciphers: the tweakey framework. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 274–288. Springer, 2014.

[JNPS16] Jérémy Jean, Ivica Nikolic, Thomas Peyrin, and Yannick Seurin. Deoxys v1. 41. *Submitted to CAESAR*, 124, 2016.

[KCP17] Mustafa Khairallah, Anupam Chattopadhyay, and Thomas Peyrin. Looting the luts: Fpga optimization of aes and aes-like ciphers for authenticated encryption. In *International Conference on Cryptology in India*, pages 282–301. Springer, 2017.

[KDT$^+$19] Jens-Peter Kaps, William Diehl, Michael Tempelmeier, Ekawat Homsirikamol, and Kris Gaj. Hardware API for lightweight cryptography. *URL https://cryptography. gmu. edu/athena/index. php*, pages 1–26, 2019.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual international cryptology conference*, pages 388–397. Springer, 1999.

[KPC20] Mustafa Khairallah, Thomas Peyrin, and Anupam Chattopadhyay. Preliminary hardware benchmarking of a group of round 2 nist lightweight aead candidates. Cryptology ePrint Archive, Report 2020/1459, 2020. https://ia.cr/2020/1459.

[KR11] Ted Krovetz and Phillip Rogaway. The Software Performance of Authenticated-Encryption Modes. In *International Workshop on Fast Software Encryption*, pages 306–327. Springer, 2011.

[KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. Silver–statistical independence and leakage verification. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 787–816. Springer, 2020.

[LBS19] Itamar Levi, Davide Bellizia, and François-Xavier Standaert. Reducing a masked implementation's effective security order with setup manipulations: And an explanation based on externally-amplified couplings. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 293–317, 2019.

[LRW02] Moses Liskov, Ronald L Rivest, and David Wagner. Tweakable Block Ciphers. In *Annual International Cryptology Conference*, pages 31–46. Springer, 2002.

[MRSS18] Amir Moradi, Bastian Richter, Tobias Schneider, and François-Xavier Standaert. Leakage detection with the χ^2-test. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 209–237, 2018.

[MV04] David McGrew and John Viega. The Galois/counter mode of operation (GCM). *submission to NIST Modes of Operation Process*, 20:0278–0070, 2004.

[rom] Romulus Authenticated Encryption / Hash. https://romulusae.github.io/romulus/specs.

[RS06] Phillip Rogaway and Thomas Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 373–390. Springer, 2006.

[SM15] Tobias Schneider and Amir Moradi. Leakage assessment methodology. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 495–513. Springer, 2015.

[SRW22] Alon Shakevsky, Eyal Ronen, and Avishai Wool. Trust Dies in Darkness: Shedding Light on Samsung's TrustZone Keymaster Design. In *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA, August 2022. USENIX Association.

[Sta18] François-Xavier Standaert. How (not) to use welch's t-test in side-channel security evaluations. In *International conference on smart card research and advanced applications*, pages 65–79. Springer, 2018.

[Tec21] Technical Committee : ISO/IEC JTC 1/SC 27 Information security, cybersecurity and privacy protection. ISO/IEC 18033-7: Information security — Encryption algorithms — Part 7: Tweakable block ciphers, 2021.

[TMÇ+19] Meltem Sönmez Turan, Kerry A McKay, Çagdas Çalik, Donghoon Chang, Lawrence Bassham, et al. Status report on the first round of the NIST lightweight cryptography standardization process. *National Institute of*

Standards and Technology, Gaithersburg, MD, NIST Interagency/Internal Rep.(NISTIR), 2019.

[TMC$^+$21] Meltem Sönmez Turan, Kerry McKay, Donghoon Chang, Cagdas Calik, Lawrence Bassham, Jinkeon Kang, John Kelsey, et al. Status report on the second round of the nist lightweight cryptography standardization process. Technical report, Technical Report, 2021.

[Wil] Stephen Williams. THE ICARUS VERILOG COMPILATION SYSTEM. https://github.com/steveicarus/iverilog.