

Sonikku: Gotta Speed, Keed!

A Family of Fast and Secure MACs

Amit Singh Bhati^{1,2}, Elena Andreeva³, Simon Müller³ and Damian Vizár⁴

¹ COSIC, KU Leuven, Belgium

² 3MI Labs, Belgium

³ TU Wien, Austria

⁴ CSEM, Switzerland.

amitsingh.bhati@3milabs.tech, {elena.andreeva,simon.mueller}@tuwien.ac.at,
damian.vizar@csem.ch

Abstract.

A message authentication code (MAC) is a symmetric-key cryptographic function used to authenticate a message by assigning it a tag. This tag is a short string that is difficult to reproduce without knowing the key. The tag ensures both the authenticity and integrity of the message, enabling the detection of any modifications.

A significant number of existing message authentication codes (MACs) are based on block ciphers (BCs) and tweakable block ciphers (TBCs). These MACs offer various trade-offs in properties, such as data processing rate per primitive call, use of single or multiple keys, security levels, pre- or post-processing, parallelizability, state size, and optimization for short/long queries.

In this work, we propose the **Sonikku** family of expanding primitive based MACs, consisting of three instances: **BabySonic**, **DarkSonic**, and **SuperSonic**. The **Sonikku** MACs are – 1) faster than the state-of-the-art TBC-based MACs; 2) secure beyond the birthday bound in the input block size; 3) smaller in state size compared to state-of-the-art MACs; and 4) optimized with diverse trade-offs such as pre/post-processing-free execution, parallelization, small footprint, and suitability for both short and long queries. These attributes make them favorable for common applications as well as “IoT” and embedded devices where processing power is limited.

On a Cortex-M4 32-bit microcontroller, **BabySonic** with **ForkSkinny** achieves a speed-up of at least 2.11x (up to 4.36x) compared to state-of-the-art ZMAC with **SKINNY** for 128-bit block sizes and queries of 95B or smaller. **DarkSonic** and **SuperSonic** with **ForkSkinny** achieve a speed-up of at least 1.93x for small queries of 95B or smaller and 1.48x for large queries up to 64KB, respectively, against ZMAC with **SKINNY** for both 64- and 128-bit block sizes.

Similar to ZMAC and PMAC2x, we then demonstrate the potential of our MAC family by using **SuperSonic** to construct a highly efficient, beyond-birthday secure, stateless, and deterministic authenticated encryption scheme, which we call **SonicAE**.

Keywords: Authentication, MAC, forkcipher, lightweight cryptography, provable security, related-tweakey, parallel, sequential, length independent security, short queries.

1 Introduction

Message Authentication Codes (MACs) are fundamental to cryptography, ensuring authenticity and integrity in two-party, secret key communication. The sender generates a tag over the intended message using a MAC with the secret key K . Upon receiving the

message-tag pair, the receiver verifies the message’s authenticity and integrity by running a MAC verification algorithm with the same secret key K .

In recent years, numerous MACs based on block cipher (BC) and tweakable block cipher (TBC) have been proposed, including the NIST standardized CMAC [26], PMAC [17], the ISO/IEC standard LightMAC [34], and their optimized successors such as PMAC1 [40], PMAC_Plus [45], 1k-PMAC_Plus [21], PMAC_TBC1k [35], and the state-of-the-art (SotA) MACs PMAC2x [32] and ZMAC [27].

Many of these MACs offer birthday-bound (BB) security guarantees, i.e., to achieve $n/2$ -bit security, a (T)BC with an n -bit block size is required. (T)BCs with small block sizes ($n = 64$) have a smaller footprint and are better suited for constrained environments when compared with larger block (T)BCs such as with $n = 128$. However, using BB-secure MACs with such a 64-bit small block size is prohibitive due to practical attacks [12]. This makes MACs with beyond BB security desirable for many lightweight platforms.

(T)BC-based MACs can be further divided into sequential and parallel processing types. Sequential MACs have a smaller footprint and often avoid post-processing/finalization call overheads, making them useful for memory-constrained applications processing predominantly short messages. Parallel MACs allow full parallelization and enjoy significant speedups with hardware acceleration on multi-core platforms. Parallel MACs can also benefit from incremental processing [9], where a change in a few message bits/blocks does not require recomputing the entire message, reducing tag re-computation costs for long messages. This property is useful when the tag is frequently generated and updated.

(T)BC-based MACs can also be divided into deterministic and nonce-based types [18]. As such, (T)BC-based MACs are important components not only for authentication but also in authenticated encryption (AE) schemes.

MACs suitable for low-cost, low-power IoT or embedded small devices, which may lack cryptographic hardware acceleration, must balance several conflicting design choices. For example, large block sizes for adequate security increase the area, while smaller state size or reduced number of TBC primitive calls with no pre-/post-processing improves performance for short message setups but may reduce security.

For instance, when messages need to be authenticated by the remote control of industrial heavy machinery, the data is typically a few bytes, consisting of a bitmap of pressed buttons and a quantized joystick position, sent by an optimized application-specific protocol. The communication round trip must meet stringent latency restrictions for safety, with an upper bound of 100 ms common in the industry [1, 42, 46]. Reducing the total computational time for tag computation is critical in such applications.

Longer messages that need authentication include large binary files verifiable at runtime, such as multimedia content for embedded devices with displays. These messages range from KB (pictures) to MB (videos). On low-end microcontrollers, content manipulation can drain most computational resources, leaving little time for MAC computation, especially without cryptographic hardware acceleration.

While an expanding primitive may seem counterintuitive for authentication, this work shows that tweakable expanding primitive (such as forkciphers [6] like ForkSkinny [38] and expanding PRFs like ButterKnife [4], ForkEDMD [23] and ForkCENC [23]) based MACs can overcome traditional barriers in lightweight MAC design. A forward forkcipher evaluation is equivalent to two parallel (T)BC calls with two independent keys at a lower cost whereas an n -to- $2n$ -bit expanding PRF is equivalent to two parallel n -to- n -bit PRFs with independent keys at a lower cost.

In literature, Forkciphers and expanding PRFs have been shown useful to construct cryptographic schemes with optimized performance and/or security of – 1) authenticated encryption with associated data (AEAD) for short messages [6] and with leakage resilience [11, 19]; 2) robust online authenticated encryption [3, 5, 13] for traditional applications; 3) transciphering-friendly AEAD schemes [15] for IoT-to-cloud use cases; 4)

efficient and beyond BB-secure confidentiality-only encryption schemes [2] and tweakable enciphering schemes [16]; 5) pseudo random number generators [7] and 6) key derivation functions [14]. For example, ForkSkinny is a forkcipher, which forks the internal SKINNY state approximately halfway through execution, provide favorable performance-security trade-offs.

Our Contribution. In this work, we propose the Sonikku MAC family based on expanding primitives. Sonikku includes two sequential MACs BabySonic and DarkSonic and one fully parallelizable MAC SuperSonic. Sonikku MACs achieve the strong notion of variable-input-length (VIL) pseudorandom function (PRF) security ranging from beyond birthday bound (BBB) to full n bits levels. DarkSonic and SuperSonic achieve the PRF security when the forkcipher is a secure pseudorandom tweakable forkcipher (prtfp) [6]. Further, under our extended security definition for forkcipher under XOR-related-tweakey setting (xrtk-prtfp), all Sonikku members achieve PRF security. All Sonics provide security *strictly* beyond the birthday bound in n , the forkcipher input block size, assuming its tweak size $t \geq n$. More precisely, DarkSonic and SuperSonic achieve close to full n -bit security and BabySonic achieves $3n/4$ -bit security. In addition, our MACs significantly improve over SotA TBC based MACs in terms of performance and flexibility. When compared to ZMAC, the Sonikku members:

1. require smaller state for a given security level.
2. require significantly lower number of rounds (at least 33% less rounds).
3. more concretely, on 32-bit Cortex-M4 processor with $n = 64$ (or 128), can save at least 40% (or 34%) number of clockcycles, respectively for any message size. For short messages of size $\leq 48\text{B}$, BabySonic with $n = 128$ can save 77% number of clockcycles.
4. come with pre-processing and post-processing optimizations, parallelization and small footprint, are optimized for short queries and long queries, etc.

We refer the reader to Table 1 for a detailed comparison of Sonikku MACs with a number of existing and relevant (T)BC-based MACs and to Fig. 5 and Table 2 for performance comparisons of Sonikku MACs instantiated with ForkSkinny with SotA ZMAC and PMAC2x instantiated with SKINNY. For simplicity, in the rest of this work, we use $\Pi[E]$ to denote the MAC function Π instantiated with the primitive E .

BabySonic[ForkSkinny] gains a speed-up of at least **2.11x** (and up to **4.36x**) against ZMAC[SKINNY] for $n = t = k = 128$ (where k is the key size) for queries of size 95B or smaller and DarkSonic[ForkSkinny] and SuperSonic[ForkSkinny] gain a speed-up of at least **1.93x** and **1.48x** against ZMAC[SKINNY] for $k = 128$ and $t = n \in \{64, 128\}$ for small queries of size 95B (or smaller) and large queries up to 64KB, respectively.

Finally, we propose an authenticated encryption with associated data (AEAD) scheme named SonicAE based on the SIV-type composition of SuperSonic and the GCTR₂₋₃ [2] encryption scheme in Sec. 6. Our SonicAE mode provides the stronger AEAD security guarantee of misuse-resistant AE (MRAE) [41]. We refer the reader to Fig. 7 and Table 3 for more details on performance estimation and comparison of SonicAE with other SotA AEAD schemes. SonicAE[ForkSkinny] gains a speed-up of at least factor 1.41x and 1.32x against existing MRAE schemes ZAE[SKINNY] and Deoxys-II[SKINNY] under $k = 128$, $t = n \in \{64, 128\}$ for small queries of size 95B and large queries up to 64KB, respectively.

Related Work. In [20], Datta et al. proposed LightFORK, a forkcipher variant of LightMAC that improves it by achieving beyond birthday security. However, in contrast to our parallel MAC – SuperSonic who can process at least $n + k$ bit per one-legged forkcipher call, LightFORK can only process n bits per two-legged forkcipher call which makes it at least **3x** costlier than SuperSonic for $k \geq n$.

Table 1: Comparison of Sonikku modes with state of the art MACs with comparable instances. Each entry referring to a binary string size or security parameter of a MAC is in bits. n , t and k are the block, tweak and key sizes of the underlying primitive, respectively whereas $e = \log_2 \left(\left\lceil \frac{\max_i m_i}{n+t+k-e} \right\rceil \right)$ and $s = \log_2 \left(\left\lfloor \frac{\max_i m_i}{n-s} \right\rfloor \right)$ are the reserved tweak bits for the counter in the corresponding MAC. BC, TBC, FC, BM, Par., L.I.Sec. and IT are short for block cipher, tweakable block cipher, forkcipher, binary (field) multiplications, parallelization support, (message) length independent security and information-theoretic security, respectively. The two columns $|K|$ and $|Tag|$ represent the key length and the maximum possible tag length of the MAC, respectively. For DarkSonic and SuperSonic, the table rows remain intact correct if the parts in blue are removed, i.e. the security still holds but at the cost of reduced performance. The cells in red highlight the key points of comparison in a MAC against Sonics.

MAC	Primitive	PRF Security (IT)	$ K $	Minimal State	Cost ¹ (to process an m -bit message)	$ Tag $	L.I.Sec.	Par.
CMAC [26]	BC as prp [31]	$n/2$	k	$3n+k$	$(\lceil m/n \rceil + 1)BC + 2BM$	$\leq n$	\times	\times
PMAC [17]	BC as prp	$n/2$	k	$3n+k$	$(\lceil m/n \rceil + 1)BC + \lceil m/n \rceil BM$	$\leq n$	\times	\checkmark
PMAC1 [40]	TBC as tprp [31]	$n/2$	k	$2n+t+k$	$\lceil m/n \rceil TBC$	$\leq n$	\times	\checkmark
lightMAC [34]	BC as prp	$n/2$	$2k$	$3n+2k$	$\lceil m/(n-s) \rceil BC$	$\leq n$	\checkmark	\checkmark
SUM-ECBC [44]	BC as prp	$2n/3$	$4k$	$2n+4k$	$2(\lceil m/n \rceil + 1)BC$	$\leq n$	\times	\times
PMAC_Plus [45]	BC as prp	$2n/3$	$3k$	$5n+3k$	$(\lceil m/n \rceil + 4)BC + (3\lceil m/n \rceil - 1)BM$	$\leq n$	\times	\checkmark
1k-PMAC_Plus [21]	BC as prp	$2n/3$	k	$5n+k$	$(\lceil m/n \rceil + 4)BC + (3\lceil m/n \rceil)BM$	$\leq n$	\times	\checkmark
ZMAC [27]	TBC as tprp	$\min\{n, (n+t)/2\}$	k	$4n+2t+k$	$\left(\left\lceil \frac{m}{\lceil n+t-s \rceil} \right\rceil + 6 \right) TBC + \left(2 \left\lceil \frac{m}{\lceil n+t-s \rceil} \right\rceil - 1 \right) BM$	$\leq 2n$	\times	\checkmark
PMAC_TBC1k [35]	TBC as tprp	n (if $t \geq n/2$)	k	$3n+t+k$	$(\lceil m/n \rceil + 2)TBC + (\lceil m/n \rceil - 1)BM$	$\leq n$	\checkmark	\checkmark
PMACx/PMAC2x [32]	TBC as tprp	n	k	$3n+t+k$	$(\lceil m/n \rceil + 2)TBC + \lceil m/n \rceil BM$	$\leq 2n$	\checkmark	\checkmark
BabySonic [Our Work]	FC as xrtk-prtfrp	$3n/4$	k	$2n+t+k$	$\left(\left\lceil \frac{m}{n+t+k-2} \right\rceil \right) FC$	$\leq 1.5n$	\times	\times
DarkSonic [Our Work]	FC as xrtk-prtfrp	$\min\{n, t\} - \log_2 \mu$	k	$2n+t+2k$	$\left(\left\lceil \frac{m-2(t+k-2)}{n+t+k-2} \right\rceil \right) TBC + 2FC$	$\leq 2n$	\times	\times
SuperSonic [Our Work]	FC as xrtk-prtfrp	$\min\{n, (n+t)/2\}$	k	$2n+2t+3k$	$\left(\left\lceil \frac{m}{n+t+k-e} \right\rceil \right) (TBC + BM) + 1FC$	$\leq 2n$	\checkmark	\checkmark

Paper Organization. In Sec. 2, we give the necessary notations and the security notions. In Sec. 3, we formally describe Sonikku and the related security results and we defer in parts the security analysis to Sec. 5. We discuss our Sonikku results and their software performances in Sec. 4. We present SonicAE as an application of SuperSonic for misuse-resistant authenticated encryption in Sec. 6. We conclude the paper in Sec. 7.

2 Preliminaries

Strings and Operations. All strings are considered as binary strings. The set of all strings of all possible lengths is denoted by $\{0, 1\}^*$ and the set of all strings of length n (a positive integer) is denoted by $\{0, 1\}^n$. We denote by $\text{Perm}(n)$ the set of all permutations of $\{0, 1\}^n$ and by $\text{Func}(m, n)$ the set of all functions with domain $\{0, 1\}^m$ and range $\{0, 1\}^n$.

For a string X of ℓ bits, we denote by $X[i]$ the i^{th} bit of X for $i = 1, \dots, \ell$ (counting from left to right) and define $X[i \dots j] = X[i] \parallel X[i+1] \parallel \dots \parallel X[j]$ for $1 \leq i < j \leq \ell$. For two strings $X, Y \in \{0, 1\}^*$ with (w.l.o.g.) $|X| \leq |Y|$, we let $X \oplus Y$ denote the bitwise XOR of $X \parallel 0^{|Y|-|X|}$ and Y . For the same strings, we define $X \oplus_a Y = (X \oplus Y)[0 \dots a - 1]$.

We fix an arbitrary integer n for this work and call it the block size. We use $X_1, \dots, X_x, X_* \stackrel{n}{\leftarrow} X$ to define the partitioning of X into n -bit blocks, such that $X = X_1 \parallel \dots \parallel X_x \parallel X_*$ with $|X_i| = n$ for $i = 1, \dots, x$ and $0 < |X_*| \leq n$. Hence, $x = \lceil |X|/n \rceil - 1$. For two distinct strings $X, Y \in \{0, 1\}^*$ with $|X| \leq |Y|$ w.l.o.g, we let $\text{lcp}_n(X, Y) = \max\{1 \leq i \leq \lceil |X|/n \rceil - 1 \mid X_j = Y_j \text{ for } 1 \leq j \leq i\}$ denote the length of the longest common prefix (in n -bit blocks) of X and Y .

Miscellaneous. We let $X \leftarrow_s \mathcal{X}$ denote the sampling of an element X from a finite set \mathcal{X} under the uniform distribution. We let $(p)_q$ denote the falling factorial $p \cdot (p-1) \cdot (p-2) \cdot \dots \cdot (p-q+1)$.

¹Actual performance cost of a FC, BC and TBC depend on their instantiations. To exemplify, when instantiated using SKINNY's round function with the same key, block and/or tweak size, 1 FC costs approx 1.6(T)BC. For concrete performance comparison, see Sec. 4.

$\dots \cdot (p - q + 1)$ where $(p)_0 = 1$. We define a predicate $P(x)$ as $P(x) = 1$ if it is true and $P(x) = 0$ if it is false. We use lexicographic comparison for integer tuples (to exemplify, $(i', j') < (i, j)$ iff $i' < i$ or $i' = i$ and $j' < j$).

2.1 MAC Syntax and Security Definition

Let \mathcal{X} and \mathcal{Y} be non-empty finite sets. Let $\text{Func}(\mathcal{X}, \mathcal{Y})$ be the set of all functions from \mathcal{X} to \mathcal{Y} . A uniform random function (URF) with domain \mathcal{X} and range \mathcal{Y} , denoted $R : \mathcal{X} \rightarrow \mathcal{Y}$, is a random function with uniform distribution over $\text{Func}(\mathcal{X}, \mathcal{Y})$.

A MAC is a tuple of function $\text{MAC} : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ with non-empty key space \mathcal{K} , and a tag verification function $\text{Verify} : \mathcal{K} \times \mathcal{X} \times \mathcal{Y} \rightarrow \{1, \perp\}$, where for all $K \in \mathcal{K}$ and $X \in \mathcal{X}$, $\text{Verify}_K(X, Y)$ returns 1 iff $\text{MAC}_K(X) = Y$ and \perp otherwise. It is well-known that if MAC is a secure PRF, it is also a secure MAC; however, the converse statement is not necessarily true. We now recall the standard security definition of PRF security for a MAC.

Definition 1 (PRF Advantage). For $\text{MAC} : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$, let \mathcal{A} be an adversary whose goal is to distinguish MAC_K and a URF $R : \mathcal{X} \rightarrow \mathcal{Y}$ by their oracle access. The advantage of \mathcal{A} against the PRF-security of MAC is then defined as

$$\text{Adv}_{\text{MAC}}^{\text{PRF}}(\mathcal{A}) = |\Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\text{MAC}_K} \Rightarrow 1] - \Pr[R \xleftarrow{\$} \text{Func}(\mathcal{X}, \mathcal{Y}) : \mathcal{A}^R \Rightarrow 1]|.$$

2.2 Tweakable Expanding Primitives

A tweakable expanding primitive $F_\alpha : \{0, 1\}^k \times \{0, 1\}^t \times \{0, 1\}^n \rightarrow \{0, 1\}^{\alpha n}$ maps a k -bit key K , t -bit tweak T and n -bit input X to $\alpha \geq 2$ many n -bit outputs $Y_1, Y_2, \dots, Y_\alpha$. Further, when all α outputs are permutations of the input, F_α is called a multiforkcipher (MFC) [2]. For this work, we set $\alpha = 2$ and hence restrict F_α to $(n + t + k)$ -to- $2n$ -bit expanding functions and forkciphers [6] (i.e., MFCs with $\alpha = 2$). We drop the subscript α now and denote an $(n + t + k)$ -to- $2n$ -bit expanding primitive by F . We call k, n and t the keysize, blocksize and tweaksize of F , respectively. We use $F(K, T, M) = F_K(T, M) = F_K^T(M)$ interchangeably.

We note that an $(n + t + k)$ -to- $2n$ -bit function may not seem expanding on its full domain, however, the term “expanding” here refers to the input space to output space expansion which is n -bit to $2n$ -bit. We use an additional parameter $s \in \{0, 1, \mathbf{b}\}$ to separate partial and full F calls as $F_K^{T,s}(M) := F_K^T(M)[1 \dots n]$ when $s = 0$, $F_K^T(M)[n + 1 \dots 2n]$ when $s = 1$ and $F_K^T(M)$ when $s = \mathbf{b}$. For the rest of the paper, we use $\mathcal{K} := \{0, 1\}^k$ and $\mathcal{T} := \{0, 1\}^t$.

XOR-related-tweakey (xrtk) Security of Tweakable Expanding Primitives. We define a strong security notion for tweakable expanding primitives that generalizes standard security notions of PRFs and forkciphers to related-tweakey setting [10]. Related-key security for block ciphers was originally proposed by Bellare and Kohno at Eurocrypt’03 [10]. Their proposed security model allows the adversary to choose a related key derivation (RKD) function ϕ which transforms the target key K into the key $\phi(K)$, and then to obtain the value of the block cipher, on an input of adversary’s choice, under this transformed key. We naturally extend this definition to expanding primitives and further restrict the relation of keys ϕ to XOR i.e. $\phi \in \Phi_k^\oplus = \{C \in \{0, 1\}^k \mid \phi_C\}$ where $\phi_C(K) = K \oplus C$. We refer the reader to [10, Corollary 1] for more details on concrete security results for XOR restricted RKDs.

Forkcipher Security (Related-tweakey Setting). Formally, we define the xrtk forkcipher security of an expanding primitive F as the indistinguishability between the real xrtk-real_F and the ideal $\text{xrtk-ideal-prtfp}_F$ worlds when adversary accesses either worlds in a chosen plaintext fashion. In the real world, the forkcipher oracle implements

Game xrtk-real_F	Game xrtk-prtfp-ideal_F	Game xrtk-prf-ideal_F
$f_1 \leftarrow \text{Func}(k, t), f_2 \leftarrow \text{Func}(k, k)$	for $T_1 \ T_2 \in \{0, 1\}^{t+k}$ do	for $T_1 \ T_2 \in \{0, 1\}^{t+k}$ do
$K \leftarrow \mathcal{K}, K_1 \leftarrow f_1(K), K_2 \leftarrow f_2(K)$	$\pi_{T_1 \ T_2, 0}, \pi_{T_1 \ T_2, 1} \leftarrow \text{Perm}(n)$	$f_{T_1 \ T_2, 0}, f_{T_1 \ T_2, 1} \leftarrow \text{Func}(n, n)$
$b \leftarrow \mathcal{A}^\varepsilon$	$b \leftarrow \mathcal{A}^\varepsilon$	$b \leftarrow \mathcal{A}^\varepsilon$
return b	return b	return b
Oracle $\mathcal{E}(T_1, T_2, X)$	Oracle $\mathcal{E}(T_1, T_2, X)$	Oracle $\mathcal{E}(T_1, T_2, X)$
return $F_{K_2 \oplus T_2}(K_1 \oplus T_1, X)$	return $\pi_{T_1 \ T_2, 0}(X), \pi_{T_1 \ T_2, 1}(X)$	return $f_{T_1 \ T_2, 0}(X), f_{T_1 \ T_2, 1}(X)$

Figure 1: xrtk security games for an expanding primitive F.

the true F algorithm faithfully but with the tweak set to $K_1 \oplus T_1$ and the key set to $K_2 \oplus T_2$ for the adversarial chosen inputs $(T_1, T_2) \in \mathcal{T} \times \mathcal{K}$ and random subkeys $K_1 := f_1(K)$ and $K_2 := f_2(K)$ for some $f_1 \leftarrow \text{Func}(k, t), f_2 \leftarrow \text{Func}(k, k)$ and $K \leftarrow \mathcal{K}$ whereas in the latter world, the oracle replaces F by two tweakable random permutations $\pi_{T_1 \| T_2, 0}, \pi_{T_1 \| T_2, 1} \leftarrow \text{Perm}(n)$ for $(T_1, T_2) \in \mathcal{T} \times \mathcal{K}$. Both worlds are depicted in Fig. 1. We then define the xrtk-prtfp advantage of \mathcal{A} as:

$$\text{Adv}_{\text{F}}^{\text{xrtk-prtfp}}(\mathcal{A}) = |\Pr[\mathcal{A}^{\text{xrtk-real}_F} \Rightarrow 1] - \Pr[\mathcal{A}^{\text{xrtk-prtfp-ideal}_F} \Rightarrow 1]|.$$

In a nutshell, under **xrtk-prtfp** notion, a forkcipher is expected to behave like a pair of random permutations that can be tweaked with two inputs T_1 and T_2 . Changing any of them under the same but secret and random key results into two freshly sampled independent random permutations. We emphasize that for a given primitive F, **xrtk-prtfp** is a stronger assumption in the sense that it allows the adversary to attempt related-tweakey attacks and is a weaker assumption in the sense that it restricts the adversary to make inverse oracle calls when compared with the standard **prtfp** [6] assumption.

Expanding PRF Security (Related-tweakey Setting). Formally, the xrtk PRF security of an expanding primitive F is defined in the exact same manner as the xrtk forkcipher security defined above but with two tweakable random permutations $\pi_{T_1 \| T_2, 0}, \pi_{T_1 \| T_2, 1} \leftarrow \text{Perm}(n)$ in the ideal world replaced with two functions $f_{T_1 \| T_2, 0}, f_{T_1 \| T_2, 1} \leftarrow \text{Func}(n, n)$ for $(T_1, T_2) \in \mathcal{T} \times \mathcal{K}$. We call this ideal world **xrtk-prf-ideal_F** and formally describe it in Fig. 1. We then define the xrtk-prf advantage of \mathcal{A} as:

$$\text{Adv}_{\text{F}}^{\text{xrtk-prf}}(\mathcal{A}) = |\Pr[\mathcal{A}^{\text{xrtk-real}_F} \Rightarrow 1] - \Pr[\mathcal{A}^{\text{xrtk-prf-ideal}_F} \Rightarrow 1]|.$$

The related-(twea)key security is the most suitable notion to work with when arguing security of feedback-based MAC designs. Key feedbacks allow MACs to achieve a state size equal to that of the underlying primitive e.g., BabySonic (see Table 1).

Relation Between Single-key and XOR-related-key Security. We emphasize that the generic results of [10][Lemma 5.3, 5.4, Theorem 6.3 and 8.8] show that any standard single-key prtfp and prf can directly be used as XOR-related-key prtfp and prf, respectively but with an additional birthday degradation term in the security bound of $p\sigma/2^{|K_1|+|K_2|}$ where p and σ are numbers of primitive calls with different adversary-chosen random keys and with the secret key XORed with adversary chosen relations/values, respectively.

For example, with ForkSkinny, we have $|K_1| + |K_2| \geq 2n$ and therefore, the security degradation from single-key to XOR-related-key setting becomes significant only after $\mathcal{O}(2^n)$ primitive calls. In other words, we have at least n bits of security for using a single-key secure primitive in XOR-related-key setting.

We also note that the xor-related-tweakey security of ForkSkinny has not been challenged by existing cryptanalysis results. The tweakable results of [8, 22, 33, 39, 47] and the related-tweakey security of SKINNY and ForkSkinny are evidences in that direction.

3 Sonikku Family of Fast and Secure MACs

Sonikku is a family of three fast and secure expanding primitive based MACs - BabySonic, SuperSonic and DarkSonic. All Sonics provably achieve BBB PRF-security and a significant speed-up compared to the SotA MACs under various application settings (see Sect. 4 for full details). To achieve a significant speed-up, all Sonics not only effectively utilize the input-space and tweak-space of the underlying primitive but also its key-space to *securely* combining message blocks with the available key material.

Design Choices. We discuss the distinct choices towards designing our concrete MACs. SuperSonic is designed with HW parallelization support and fast performance for long queries in mind. Towards these goals, it uses parallel primitive calls to process the message and maintains an extra state to store the chained internal values for the final F call a.k.a. the finalization call of the MAC (see Fig. 3 for details). Compared to SuperSonic, the other two MACs - DarkSonic and BabySonic trade the HW parallelization support for reducing the minimal state size (which is a part of the required RAM). Hence, they are more suitable for applications where parallelization is not supported or where resource constraints such as small and fixed state size are present.

BabySonic uses the key only once during the initialization, and hence it avoids the need to store the key. As a consequence, BabySonic achieves the optimal minimal-state size i.e., same as the state of the underlying expanding primitive. BabySonic also uses both branches of F to generate an internal pseudorandom state and uses that to eliminate pre-processing and post-processing calls such as the first F call in DarkSonic and the last F call in SuperSonic which makes it faster and optimized for short queries up to 95B.

On the other hand, DarkSonic makes use of a nonce input and eliminates the need of computing the second leg in F calls. This way it achieves significantly better performance with increasing message length (close to SuperSonic for queries of any size) when compared to BabySonic but with a nonce-based security (see Table 1 for full details).

Our Pick. We pick BabySonic (for short messages of size $< 136\text{B}$, see Fig. 5(b)) followed by SuperSonic (for long queries of size $> 96\text{B}$, see Fig. 5(b)) as our best choices and keep DarkSonic as a bonus MAC with intermediate trade-offs.

In the following sections, we formally define each of the Sonics and state their PRF security results. Our security results are backed-up with concrete mathematical proofs.

3.1 BabySonic_{n/a} and its PRF Security

BabySonic_{n/a} is a sequential MAC that uses a tweakable expanding primitive F (as defined in Sec. 2.2) as an underlying primitive with $\mathcal{T} = \{0, 1\}^t$ for positive integers t . BabySonic_{n/a}[F] = (\mathcal{K} , MAC) has a key space $\mathcal{K} = \{0, 1\}^k$ and message space $\mathcal{M} = \{0, 1\}^*$. It provides “*optimal*” internal state-size which is equal to the minimal argument-size of an expanding primitive (in other words, theoretically there is no overhead in the state when shifting from the cipher to the MAC). The MAC algorithm of BabySonic_{n/a} is illustrated in Fig. 2 and pseudocode of the same is provided in Fig. 8. We state the formal claim about the PRF security of BabySonic_{n/a} in Theorem 1 and provide its proof in Sec. 5.1.

Theorem 1. *Let F be a tweakable expanding primitive with $\mathcal{T} = \{0, 1\}^t$ and $k \leq n + a$ for some integer $1 \leq a \leq n$. Then for any adversary \mathcal{A} who makes at most q queries (with maximum query length of ℓ blocks) to the BabySonic_{n/a} MAC such that the total number of*

primitive calls induced by all the queries is at most σ and $10 \leq a \leq \min\{n - 10, 11n/12\}$, we have

$$\text{Adv}_{\text{BabySonic}_{n/a}[\text{F}]}^{\text{PRF}}(\mathcal{A}) \leq \ell \cdot \text{Adv}_{\text{F}}^{\text{xrtk-prtfp}}(\mathcal{B}) + \frac{(\sigma - q)^2}{2^{n+a+1}} + \frac{6\sigma}{2^{n-a/2}}$$

for some adversary \mathcal{B} , making at most q many F queries (under fixed but secret and random key), and running in time given by the running time of \mathcal{A} plus $\gamma \cdot x$ where γ is the runtime of an F call in the model of computation.

Clearly, from the bound above one can notice that $\text{BabySonic}_{n/a}$ provides maximum (information theoretic) PRF security of $\approx 3n/4$ bits (in terms of primitive queries) when $a = n/2$ and hence, BabySonic_2 turns out to be the best variant of $\text{BabySonic}_{n/a}$. For brevity, we drop the subscript and refer this variant by BabySonic .

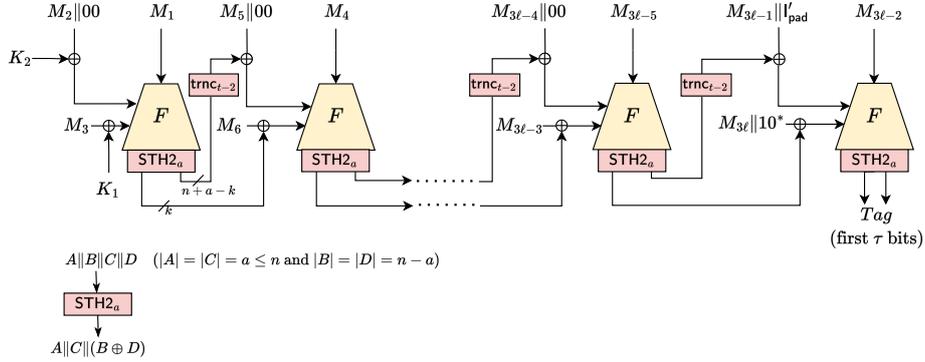


Figure 2: The $\text{BabySonic}_{n/a}$ MAC mode. The picture illustrates the processing of message M of size $x(n + t + k - 2)$ for some positive integer x . We note that for arbitrary messages of any length, a padding of 10^* is used to make the message size $x'(n + t + k - 2)$ for the smallest possible positive integer x' . Further, in this block diagram, for the last processed message block, $1'_{\text{pad}}$ is set to 01 if the size of M is $x(n + t + k - 2)$ for some positive integer x and to 11, otherwise. The small function diagram of STH2 (short for Summation-Truncation Hybrid-2 [25]) shown in the lower half of this figure shows how the $2n$ -bit outputs of F are processed to generate the $n + a$ bits of internal chaining value (or tag) where $n + a \geq k$. The MAC key K of k bits is used (with a key derivation function) to generate the sub-keys K_1 and K_2 here of size k and $t - 2$ bits, respectively.

3.2 SuperSonic and its PRF Security

SuperSonic is a parallel MAC that uses a tweakable expanding primitive F (as defined in Sec. 2.2) as an underlying primitive with $\mathcal{T} = \{0, 1\}^t$ for positive integers t . $\text{SuperSonic}[\text{F}] = (\mathcal{K}, \text{MAC})$ has a key space $\mathcal{K} = \{0, 1\}^k$ and message space $\mathcal{M} = \{0, 1\}^*$. The MAC algorithm of SuperSonic is illustrated in Fig. 3 and pseudocode of the same is provided in Fig. 8.

We state the formal claim about the PRF security of SuperSonic in Theorem 2 and provide its proof in Sec. 5.2.

Theorem 2. *Let F be a tweakable expanding primitive with $\mathcal{T} = \{0, 1\}^t$. Then for any adversary \mathcal{A} who makes at most q queries to the SuperSonic MAC such that the total number of primitive calls induced by all the queries is at most σ , we have*

$$\text{Adv}_{\text{SuperSonic}[\text{F}]}^{\text{PRF}}(\mathcal{A}) \leq \text{Adv}_{\text{F}}^{\text{xrtk-prtfp}}(\mathcal{B}) + \frac{4 \cdot q^2}{2^{n+\min\{n, t-2\}}}$$

for some adversary \mathcal{B} , making at most σ queries, and running in time given by the running time of \mathcal{A} plus $\gamma \cdot \sigma$ where γ is the runtime of an F call in the model of computation.

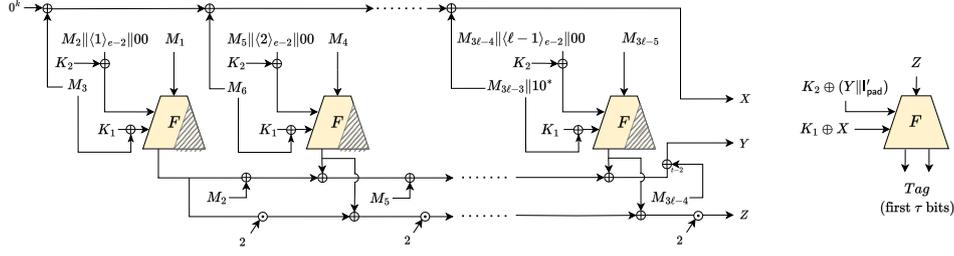


Figure 3: The SuperSonic MAC mode. The picture illustrates the processing of message M of size $x(n + t + k - e)$ for some positive integers x and a pre-defined constant e . We note that for arbitrary messages of any length, a padding of 10^* is used to make the message size the smallest possible multiple of $(n + t + k - e)$ (See Fig. 8 for details). Further, in this block diagram, for the last processed message block, l'_{pad} is set to 01 if the size of M is $x(n + t + k - e)$ for some positive integer x and to 11, otherwise. The MAC key K of k bits is used (with a key derivation function) to generate the sub-keys K_1 and K_2 here of size k and $t - 2$ bits, respectively.

3.3 DarkSonic and its PRF Security

DarkSonic is a nonce-based sequential MAC that uses a tweakable expanding primitive F (as defined in Sec. 2.2) as an underlying primitive with $\mathcal{T} = \{0, 1\}^t$ for positive integers t . $\text{DarkSonic}[F] = (\mathcal{K}, \text{MAC})$ has a key space $\mathcal{K} = \{0, 1\}^k$, nonce space $\mathcal{N} = \{0, 1\}^n$ and message space $\mathcal{M} = \{0, 1\}^*$. The MAC algorithm of DarkSonic is illustrated in Fig. 4 and pseudocode of the same is provided in Fig. 8. We state the formal claim about the PRF security of DarkSonic in Theorem 3 and provide its proof in Sec. 5.3.

Theorem 3. *Let F be a tweakable expanding primitive with $\mathcal{K} = \{0, 1\}^k$ and $\mathcal{T} = \{0, 1\}^t$. Then for any nonce-misusing adversary \mathcal{A} who makes at most q queries to the DarkSonic nonce-based MAC such that the total number of primitive calls induced by all the queries is at most σ and the maximum number of times a nonce is repeated is μ , we have*

$$\text{Adv}_{\text{DarkSonic}[F]}^{\text{PRF}}(\mathcal{A})$$

$$\leq \text{Adv}_{\mathbb{F}}^{\text{rtk-prtfp}}(\mathcal{B}) + \frac{6(\sigma - q)^2}{2^{n + \min\{n, t-2\}}} + \frac{2q(\mu - 1)}{2^{\min\{n, t-2\}}}$$

for some adversary \mathcal{B} , making at most σ queries, and running in time given by the running time of \mathcal{A} plus $\gamma \cdot \sigma$ where γ is the runtime of an F call in the model of computation.

Novelty. Below we elaborate on the novelty elements of the Sonikku MACs.

- **BabySonic** - while the design is intuitive, none of the existing MACs uses the key in a feedback manner to avoid its storage, processes at rate 1 and achieves beyond BB security. Nested-MAC (NMAC) [24] comes closest with a rate of $(n + t)/(n + t + k)$, yet it requires a FIL-PRF primitive and achieves security below BB (in k). Further, due to the key-feedback property of **BabySonic**, we need a dedicated proof where the information-theoretic counterpart of the MAC is defined through iterative replacements of the FC calls with pairs of tweakable random permutations (TRPs) (see Sec. 5.1 for details).
- **DarkSonic** - the design is the first nonce-based efficient MAC (with nonce-misuse security) that is solely based on (T)BCs or FCs. It is the first MAC that achieves full n -bit security with only n bits of chaining state (compared to the standard $2n$ bits). This improvement is achieved by securely reusing the randomness of the previous n -bit state instead of an extra n -bit state. The analysis of DarkSonic captures all possible bad cases that can happen due to this new design strategy with the use of H-coefficient's technique.

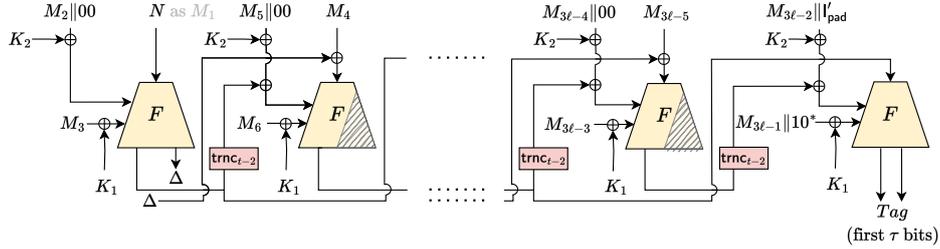


Figure 4: The DarkSonic MAC mode. The picture illustrates the processing of a full message M (including the n -bit nonce N) of size $x(n + t + k - 2) - n$ for some positive integer x . We note that for arbitrary messages of any length, a padding of 10^* is used to make the message size $x'(n + t + k - 2) - n$ for the smallest possible positive integer x' (See Fig. 8 for details). Further, in this block diagram, for the last processed message block, 10^* is set to 01 if the size of M is $x(n + t + k - 2) - n$ for some positive integer x and to 11, otherwise. Here for any arbitrary string $X \in \{0, 1\}^*$, the function $\text{trnc}_{t-2}(X)$ returns $X||0^{t-2}[1 \dots t-2]$. The MAC key K of k bits is used (with a key derivation function) to generate the sub-keys K_1 and K_2 here of size k and $t - 2$ bits, respectively.

- **SuperSonic** - SuperSonic follows the common beyond BB-secure parallel MAC design strategy as used by many popular existing MACs such as PMAC_Plus, 1k-PMAC_Plus, PMACx/2x, and ZMAC. The design novelty here lies in: 1) existing designs and proof analyses do not directly allow passing message blocks into the key argument (XORed with the key). SuperSonic efficiently achieves this by maintaining a checksum of these extra message blocks and processing it in the finalization call. 2) SuperSonic comes with a redefined efficient finalization call when compared with the SotA ZMAC. More concretely, we replace the four final TBC calls with one FC call which is relevant for short query applications. 3) SuperSonic uses strategies of “counter-in-tweak” and tweak domain separation to keep the design simple, less error-prone and maintains smaller long-term secret material which can potentially reduce the cost of threshold implementations. We give a very “compact” and “simple” n -bit security proof with “length-independent” bound when compared with the existing related MACs results.
- DarkSonic and SuperSonic can also be instantiated with TBCs instead of FCs and the security results are transferred from related-key to single-key setting without any changes to the MAC designs, except by fixing all the message blocks in the key arguments to 0s.
- Switching to single-key setting reduces the performance of Sonikku MACs, yet even after this step the speed-ups of DarkSonic and SuperSonic against ZMAC remain significant. More concretely, comparing against Table 2, we obtain that DarkSonic and SuperSonic (with $n = 128$) in single-key setting still reach up to 1.6x and 1.77x speed-ups for messages of length $\leq 95\text{B}$, respectively and these speed-ups converge close to 1x when the message lengths are very long (64KB or more).

4 Performance and Discussion

For the performance evaluation of Sonics, we used a 32-bit Cortex-M4 processor. Our proposed fully parallelizable MAC SuperSonic when compared with ZMAC provides length independent security up to full n bits and when instantiated with ForkSkinny [6] (and ZMAC with SKINNY [8]) with $k = 128$ bits and $t = n \in \{64, 128\}$, reduces the computational cost

of computing an authentication tag by a factor of 2.76x, 1.93x and 1.58x for $n = 64$ bits and of 2.66x, 2.15x and 1.48x for $n = 128$ bits when used for messages with maximum length of 16B (useful in IoT applications and embedded devices), 95B (the maximum authenticated message size in Noise framework [37] based protocols) and any longer length up to 64KB, respectively. On top of that, **SuperSonic** requires up to $2n$ -bit smaller minimal-state than ZMAC.

Adding to that, when compared with PMAC_TBC1k and PMACx/PMAC2x schemes, **SuperSonic** processes $\approx n + t + k$ bits of inputs per FC call whereas both of these compared schemes are limited to n bits of inputs per TBC call independent of the tweak/key size. We refer the reader to Table 1 for a rich comparison of **Sonikku** modes with existing state of the art MACs.

The two sequential MACs **DarkSonic** and **BabySonic** can be well-suited for applications where parallelization is not supported or resource constraints such as small and fixed statesize are present. On one hand, we have **BabySonic** that uses the key only once during the initialization and, hence avoids the need of extra state for storing the key and achieves the optimal minimal-state size (i.e. minimal-state same as the minimal-state of the underlying primitive itself) whereas on the other, we have **DarkSonic** that makes use of a nonce input and eliminates the need of computing the second leg in FC calls. This way it achieves significantly better performance with a slightly traded but comparable security.

We recall that in the work of [27], ZMAC was advertised as the first TBC-based MAC which has optimal efficiency/rate as it can process $n + t$ bits of data per TBC call as an n -bit block, t -bit tweak and k -bit key TBC was assumed to handle no more than $n + t$ bits of public input per call. In this work, we try to exploit the k -bit key argument space of a tweakable primitive (particularly, FC) in each one of the three **Sonikku** MACs to incorporate an extra k -bit of public input per call and hence resetting the margin for actual optimality.

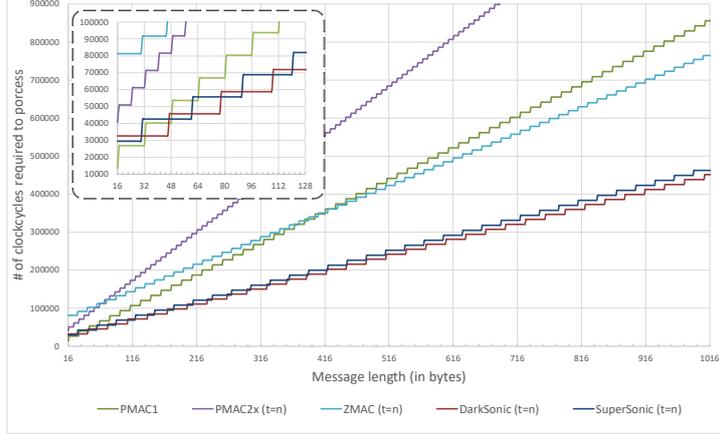
We note that passing public inputs to the key argument of a traditional (tweakable) cipher is not in general a good idea as it can decrease both the security (e.g. with related-key attacks) and efficiency (e.g. cost of re-key scheduling per call).

While key schedules are routinely precomputed to improve performance in software, many (T)BCs targeting constrained environments, such as **SKINNY**, feature lightweight key schedules where the effect of precomputation brings little benefit. In hardware, the key schedule can be computed “for free” thanks to HW parallelism. Constructions relying on frequent rekeying thus do not suffer from a performance penalty when implemented in HW and/or when using (T)BCs with a lightweight key schedule.

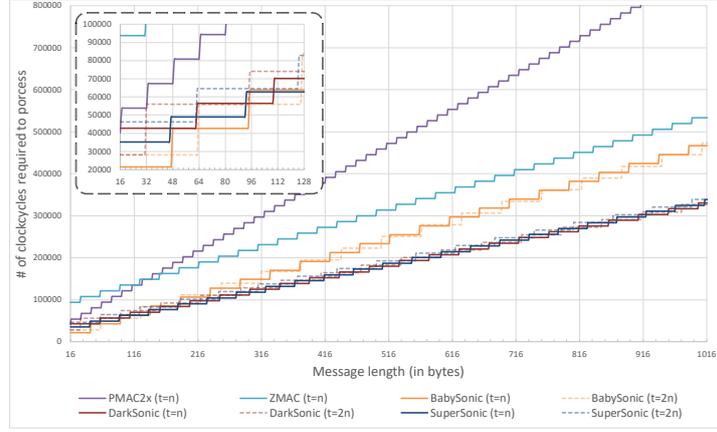
Adding to that, there exist concrete security analyses and results such as in [8, 22, 33, 39, 47] for the tweakable framework [28] and for the related-tweakey security of **SKINNY** [8] and **ForkSkinny** [6] that reasonably motivates this direction of research.

In Fig. 5, we provide an efficiency comparison of Sonic MACs (instantiated with **ForkSkinny**) with the existing state-of-the-arts (SotA) TBC-based MAC designs (instantiated with **SKINNY**) under the setting $n = k = 128$. The plot shows that the number of clockcycles required (on a typical 32-bit Cortex-M4 processor) for SotA MACs like PMAC2x or ZMAC is quite higher when compared with Sonic MACs. Among Sonics, **DarkSonic** and **SuperSonic** require almost similar number of rounds and are better than **BabySonic** for longer message queries as they make one-legged FC calls. But for small message queries **BabySonic** performs better as it does not require pre- or post-processing primitive calls. To confirm these claims, we provide a full detailed comparison of **Sonikku** MACs against ZMAC in Table 2 under both $n = 64$ and $n = 128$ settings.

We also infer from our data that using a longer tweak can improve the performance of a Sonic MAC by a significant margin but only after a certain threshold of minimum message length. For Cortex-M4, this threshold is ≈ 2 KB.



(a) MACs targeting 64 bits of security



(b) MACs targeting 96 to 128 bits of security

Figure 5: Efficiency comparison of Sonics with other (tweakable) primitive based SotA MACs under setting $k = 128$ and $n = 64$ or 128 bits (depending on the targeted security). All TBCs and FCs are instantiated with SKINNY and ForkSkinny, respectively. The plot shows the number of clockcycles that are required to process a message of corresponding length.

5 Security Analysis of Sonikku MACs

5.1 BabySonic $_{n/a}$: Proof of Theorem 1

Let us recall the alternative representation of PRF security through indistinguishability games, which is equivalent with the notion introduced in Sec. 2.1. We define two games, PRF-real $_{\Pi}$ and PRF-ideal $_{\Pi}$. In both games \mathcal{A} is given access to an oracle Π . In the game PRF-real $_{\Pi}$ the oracle Π faithfully implement the corresponding MAC algorithm of BabySonic $_{n/a}$ using a randomly sampled secret key whereas in the game PRF-ideal $_{\Pi}$, the oracle returns a uniformly sampled random string as the tag for each distinct message query. With this representation, we have that $\text{Adv}_{\Pi}^{\text{PRF}}(\mathcal{A}) = |\Pr[\mathcal{A}^{\text{PRF-real}_{\Pi}} \Rightarrow 1] - \Pr[\mathcal{A}^{\text{PRF-ideal}_{\Pi}} \Rightarrow 1]|$ or $\stackrel{\text{in short}}{=} |\Pr[\mathcal{A}^{\text{PRF-real}_{\Pi}}] - \Pr[\mathcal{A}^{\text{PRF-ideal}_{\Pi}}]|$.

Recursively replacing F. We first replace the first $F_{K_2 \oplus T_2}(K_1 \oplus T_1, \cdot)$ with a pair of independent random tweakable permutations $\pi_0 = (\pi_{T_1 \| T_2, 0} \leftarrow \$ \text{Perm}(n))_{T_1 \| T_2 \in \{0,1\}^{t+k}}$ and $\pi_1 = (\pi_{T_1 \| T_2, 1} \leftarrow \$ \text{Perm}(n))_{T_1 \| T_2 \in \{0,1\}^{t+k}}$ and let BabySonic $_{n/a}'[\text{STH2}[(\pi_0, \pi_1)], \text{STH2}[F], \dots, \text{STH2}[F]]$ denote the BabySonic mode that uses

Table 2: Efficiency comparison of Sonics (instantiated with ForkSkinny) against ZMAC (instantiated with SKINNY) under settings $k = 128$ bits and $t = n \in \{64, 128\}$ bits. The entries in this table show the ratio of number of clockcycles required by ZMAC and by corresponding Sonikku MAC (under same values of n, t and k) to process a message of the given maximum length. Here **green** and **red** cells represent settings where the corresponding Sonic is better and worse than ZMAC, respectively.

MAC →	BabySonic	DarkSonic		SuperSonic	
Block size n (in bits) →	128	64	128	64	128
IT PRF Security (in bits) →	96	64	128	64	128
Max. message length ↓	Speed-up (in factors) against ZMAC				
16B	4.36x	2.49x	2.19x	2.76x	2.66x
95B	2.11x	2.26x	2.39x	1.93x	2.15x
4KB	1.02x	1.66x	1.56x	1.60x	1.51x
64KB	0.98x	1.64x	1.52x	1.58x	1.48x

(π_0, π_1) in place of the first F call, which yields $\text{Adv}_{\text{BabySonic}_{n/a}^{\text{PRF}}}(\mathcal{A}) \leq \text{Adv}_{\text{F}}^{\text{xrtk-prtftp}}(\mathcal{B}_{q_1}) + \text{Adv}_{\text{BabySonic}'_{n/a}^{\text{PRF}}}[\text{STH2}[(\pi_0, \pi_1)], \text{STH2}[\text{F}], \dots, \text{STH2}[\text{F}]](\mathcal{A})$ where $q_i \leq q$ denotes the number of queries made by \mathcal{A} that contain at least i many primitive calls. In other words, $\sum_{i=1}^{\ell} q_i = \sigma$ with $\ell = \max\{\ell^1, \dots, \ell^q\}$ and ℓ^i as the length of the i^{th} query (in $n + t + k$ bit blocks). Here \mathcal{B}_{q_i} is an **xrtk-prtftp**-adversary against F that can make at most q_i queries to F.

Let us now further replace the first (π_0, π_1) call and its following STH2 call together with a random function $f = (f_{\text{T}_1 \parallel \text{T}_2} \leftarrow \text{Func}(n, n+a))_{\text{T}_1 \parallel \text{T}_2 \in \{0,1\}^{t+k}}$, denote it by $\text{BabySonic}'_{n/a}[f, \text{STH2}[\text{F}], \dots, \text{STH2}[\text{F}]]$ and apply the summation-truncation results from [25, Theorem 2]. This gives us for $0 \leq a \leq \min\{n-10, 11n/12\}$,

$$\begin{aligned} & \text{Adv}_{\text{BabySonic}'_{n/a}^{\text{PRF}}}[\text{STH2}[(\pi_0, \pi_1)], \text{STH2}[\text{F}], \dots, \text{STH2}[\text{F}]](\mathcal{A}) \\ & \leq \text{Adv}_{\text{BabySonic}'_{n/a}^{\text{PRF}}}[f, \text{STH2}[\text{F}], \dots, \text{STH2}[\text{F}]](\mathcal{A}) + 3 \left(\frac{q_1}{2^{n-a/3}} \right)^{3/2} + \frac{1}{\sqrt{2\pi}} \left(\frac{q_1}{2^{n-5}} \right)^{2^{n-a-2}} + \frac{\sqrt{2}q_1}{2^{n-a/2}} \\ & \leq \text{Adv}_{\text{BabySonic}'_{n/a}^{\text{PRF}}}[f, \text{STH2}[\text{F}], \dots, \text{STH2}[\text{F}]](\mathcal{A}) + \frac{6q_1}{2^{n-a/2}} \text{ when } a \geq 10, q \leq 2^{n-a/2}. \end{aligned}$$

We note that at this step the key input, let say K_2^i , to the second primitive call (or the current first FC call) in i^{th} query of $\text{BabySonic}'_{n/a}[f, \text{STH2}[\text{F}], \dots, \text{STH2}[\text{F}]]$ can be written as $(K_2^i \oplus K) \oplus K$ where $(K_2^i \oplus K)$ is independent of K as K_2^i is originally sampled independently and uniformly at random (using f). Hence, we can follow the same steps as above for this second primitive call and iterate this procedure until the last primitive call (in the longest query). Combining the bounds from all these steps, we get for $10 \leq a \leq \min\{n-10, 11n/12\}$ and $q \leq 2^{n-a/2}$,

$$\begin{aligned} \text{Adv}_{\text{BabySonic}_{n/a}^{\text{PRF}}}(\mathcal{A}) & \leq \sum_{i=1}^{\ell} \text{Adv}_{\text{F}}^{\text{xrtk-prtftp}}(\mathcal{B}_{q_i}) + \frac{6 \sum_{i=1}^{\ell} q_i}{2^{n-a/2}} + \text{Adv}_{\text{BabySonic}'_{n/a}^{\text{PRF}}}[f, \dots, f](\mathcal{A}) \\ & \leq \ell \cdot \text{Adv}_{\text{F}}^{\text{xrtk-prtftp}}(\mathcal{B}_q) + \frac{6\sigma}{2^{n-a/2}} + \text{Adv}_{\text{BabySonic}'_{n/a}^{\text{PRF}}}[f, \dots, f](\mathcal{A}). \quad (1) \end{aligned}$$

Note that since $\sigma \geq q$, for $q > 2^{n-a/2}$ this bound becomes void and hence the assumption of $q \leq 2^{n-a/2}$ can be dropped. Now, the adversary is left with the goal of distinguishing between the games $\text{PRF-real}_{\text{BabySonic}'_{n/a}[f, \dots, f]}$ and $\text{PRF-ideal}_{\text{BabySonic}'_{n/a}[f, \dots, f]}$. We note that these games are indistinguishable if there are no collisions among the inputs of f calls that correspond to non-prefixed message blocks in $\text{BabySonic}'_{n/a}[f, \dots, f]$ over q queries. We also note that there are total σ many f calls (taking corresponding F's (X, T_1, T_2) triplet

as input) out of which q calls correspond to the domain separated last f calls in q queries. Now, since each non-prefixed internal state (of $n + a$ -bit) is sampled uniformly at random using a random function, we have $\text{Adv}_{\text{BabySonic}'_{n/a}[f,\dots,f]}^{\text{PRF}}(\mathcal{A}) \leq ((\sigma^{-q}) + \binom{q}{2} - \binom{q}{2})/2^{n+a}$ where the subtracted term corresponds to the $\binom{q}{2}$ pairs of input-tweak pairs that are the first distinct input-tweak pairs in their corresponding queries (and hence there will be no collisions among them). Or we have for $10 \leq a \leq \min\{n - 10, 11n/12\}$,

$$\text{Adv}_{\text{BabySonic}_{n/a}[\mathbb{F}]}^{\text{PRF}}(\mathcal{A}) \leq \ell \cdot \text{Adv}_{\mathbb{F}}^{\text{xrtk-prtftp}}(\mathcal{B}_q) + \frac{(\sigma - q)^2}{2^{n+a+1}} + \frac{6\sigma}{2^{n-a/2}}$$

and hence the result of Theorem 1. \square

5.2 SuperSonic: Proof of Theorem 2

We use the same definition of $\text{Adv}_{\Pi}^{\text{PRF}}(\mathcal{A})$ for a MAC Π as defined in Sec. 5.1.

Replacing \mathbb{F} . We first replace $\mathbb{F}_{K_2 \oplus \mathbb{T}_2}(K_1 \oplus \mathbb{T}_1, \cdot)$ with a pair of independent random tweakable permutations $\pi_0 = (\pi_{\mathbb{T}_1 \parallel \mathbb{T}_2, 0} \leftarrow \$ \text{Perm}(n))_{\mathbb{T}_1 \parallel \mathbb{T}_2 \in \{0,1\}^{t+k}}$ and $\pi_1 = (\pi_{\mathbb{T}_1 \parallel \mathbb{T}_2, 1} \leftarrow \$ \text{Perm}(n))_{\mathbb{T}_1 \parallel \mathbb{T}_2 \in \{0,1\}^{t+k}}$ and let $\text{SuperSonic}[(\pi_0, \pi_1)]$ denote the SuperSonic mode that uses π_0, π_1 instead of \mathbb{F} , which yields $\text{Adv}_{\text{SuperSonic}[\mathbb{F}]}^{\text{PRF}}(\mathcal{A}) \leq \text{Adv}_{\mathbb{F}}^{\text{xrtk-prtftp}}(\mathcal{B}) + \text{Adv}_{\text{SuperSonic}[(\pi_0, \pi_1)]}^{\text{PRF}}(\mathcal{A})$.

Now, the adversary is left with the goal of distinguishing between the games $\text{PRF-real}_{\text{SuperSonic}[(\pi_0, \pi_1)]}$ and $\text{PRF-ideal}_{\text{SuperSonic}[(\pi_0, \pi_1)]}$. For simplicity, we denote these games by “real world” and “ideal world”, respectively. Hence, we want to bound $\text{Adv}_{\text{SuperSonic}[(\pi_0, \pi_1)]}^{\text{PRF}}(\mathcal{A}) = |\text{Pr}[\mathcal{A}^{\text{PRF-real}_{\text{SuperSonic}[(\pi_0, \pi_1)]}}] - \text{Pr}[\mathcal{A}^{\text{PRF-ideal}_{\text{SuperSonic}[(\pi_0, \pi_1)]}}]|$.

We now recall the notation that for the i^{th} query to the MAC oracle with input M^i and output Tag^i , SuperSonic (with a provided integer e) first adds padding of 10^* in the end of M^i and then internally processes the updated M^i in blocks $P_1^i, \dots, P_{\ell^i-1}^i$ (as defined in the MAC algorithm of SuperSonic, Fig. 8). Here $\ell^i - 1$ represents, the length of M^i in $(n+t+k-e)$ -bit blocks. SuperSonic also processes and XORs the internal chaining values as the accumulated randomness to its last primitive call which we denote here by $(\Delta_1^i, \Delta_2^i, \Delta_3^i)$ s where Δ_1^i, Δ_2^i and Δ_3^i correspond to the final chaining values used in (or XORed to) the final primitive call’s input, tweak \mathbb{T}_1 and tweak \mathbb{T}_2 , respectively. Let us denote the internally processed π_0 calls’ outputs as α_j^i s (corresponding to $P_j^i = M_{3j-2}^i \parallel M_{3j-1}^i \parallel M_{3j}^i$) then we can define Δ s as $\Delta_1^i = \bigoplus_{j=1}^{\ell^i-1} 2^{\ell^i-j} \alpha_j^i$, $\Delta_2^i = \bigoplus_{j=1}^{\ell^i-1} (\alpha_j^i \oplus M_{3j-1}^i)$ and $\Delta_3^i = \bigoplus_{j=1}^{\ell^i-1} M_{3j}^i$. Let us now define the bad events when the real world can be distinguished from the ideal world.

Bad \mathbb{T}_1 a.k.a. “Final Input Collision in Real-world”: There exists a pair of queries $1 \leq i' < i \leq q$ such that, the (i, ℓ^i) block call has tweak-input collision with the $(i', \ell^{i'})$ block call, i.e., $\mathbb{T}_{\ell^i}^i = \mathbb{T}_{\ell^{i'}}^{i'}$ and $\Delta_2^i = \Delta_2^{i'}$.

Bad \mathbb{T}_2 a.k.a. “Output Collision in Ideal-world”: There exists a pair of queries $1 \leq i' < i \leq q$ such that, they have same tag in the ideal-world and in real-world, the (i, ℓ^i) block call has tweak collision with the $(i', \ell^{i'})$ block call given that the inputs to these calls are distinct, i.e., in real world we have $\mathbb{T}_{\ell^i}^i = \mathbb{T}_{\ell^{i'}}^{i'}$, $\Delta_2^i \neq \Delta_2^{i'}$ and in ideal world we have $(\text{Tag}_1^i = \text{Tag}_1^{i'}) \vee (\text{Tag}_2^i = \text{Tag}_2^{i'})$.

Note that the last condition in Bad \mathbb{T}_2 of ideal-world tag collision is independent of the other two real-world conditions. Let us define Bad \mathbb{T}_{2r} (resp., Bad \mathbb{T}_{2i}) as Bad \mathbb{T}_2 with the ideal-world tag collision condition (resp., the other two real-world conditions) removed.

Now, it is easy to see that for $\Pi_{\text{ss}} = \text{SuperSonic}[(\pi_0, \pi_1)]$ we have $\text{Adv}_{\Pi_{\text{ss}}}^{\text{PRF}}(\mathcal{A})$

$$\begin{aligned}
&\leq \Pr[\text{BadT}_1] + |\Pr[\mathcal{A}^{\text{PRF-real}\Pi_{\text{ss}}} \wedge \neg \text{BadT}_1] - \Pr[\mathcal{A}^{\text{PRF-ideal}\Pi_{\text{ss}}} \wedge \neg \text{BadT}_1]| \\
&= \Pr[\text{BadT}_1] + |\Pr[\mathcal{A}^{\text{PRF-real}\Pi_{\text{ss}}} \wedge \text{BadT}_{2r}] - \Pr[\mathcal{A}^{\text{PRF-ideal}\Pi_{\text{ss}}} \wedge \text{BadT}_{2r}]| \\
&= \Pr[\text{BadT}_1] + \Pr[\text{BadT}_{2r}] \cdot |\Pr[\mathcal{A}^{\text{PRF-real}\Pi_{\text{ss}}} | \text{BadT}_{2r}] - \Pr[\mathcal{A}^{\text{PRF-ideal}\Pi_{\text{ss}}} | \text{BadT}_{2r}]| \\
&\leq \Pr[\text{BadT}_1] + \Pr[\text{BadT}_{2r}] \cdot \Pr[\text{BadT}_{2i}] = \Pr[\text{BadT}_1] + \Pr[\text{BadT}_2]. \tag{2}
\end{aligned}$$

Distribution of Δ s. We can notice that Δ_{3s} are deterministic and defined using public input, however, the other two Δ values are sampled using random tweakable permutations. We now define the exhaustive set of possible cases on how these Δ s are sampled in the i^{th} query (for any $1 \leq i \leq q$) to define an upper bound on their sampling probability.

1. For given i^{th} query, if there exists at least 2 underlying block calls (each processing input of size $n + t + k - e$ bits) that contain unique input when compared with any of the previous queries $i' < i$ s: Clearly then there are at least two α values that are fresh outputs of random permutations in both Δ_1 and Δ_2 equations and hence for any $c_1, c_2 \in \{0, 1\}^n$ we have $\Pr[\Delta_1^i = c_1 \wedge \Delta_2^i = c_2] = \Pr[\Delta_1^i = c_1] \cdot \Pr[\Delta_2^i = c_2] \leq 1/(2^n - q)^2$.
2. For given i^{th} query, if there exists only one underlying block call (processing input of size $n + t + k - e$ bits) that contains unique input when compared with some previous query $i' < i$ s: Now, there is only one α value that is fresh output of a random permutation in both Δ_1 and Δ_2 equations and hence for any $c_1, c_2 \in \{0, 1\}^n$ we have $\Pr[\Delta_1^i = c_1 \wedge \Delta_2^i = c_2] = \Pr[\Delta_1^i = c_1] \cdot \Pr[\Delta_2^i = c_2 | \Delta_1^i = c_1] \leq (1/(2^n - q)) \cdot \Pr[\Delta_2^i = c_2 | \Delta_1^i = c_1]$.

Note that since query i and i' share all except one block input, lets say a^{th} one (P_a^i), we have that $\Delta_1^i \oplus \Delta_1^{i'} = 2^{\ell-a}(\Delta_2^i \oplus \Delta_2^{i'} \oplus M_{3a-1}^i \oplus M_{3a-1}^{i'})[1 \dots n]$ which means if $c_1 = \Delta_1^{i'}$ and $c_2 = \Delta_2^{i'}$ then one of the following two always holds: 1. $\Delta_3^i \neq \Delta_3^{i'}$ and $\Pr[\Delta_1^i = c_1 \wedge \Delta_2^i = c_2] \leq (1/(2^n - q)) \cdot 1 = 1/(2^n - q)$. and 2. $\Delta_3^i = \Delta_3^{i'}$ and $\Pr[\Delta_1^i = c_1 \wedge \Delta_2^i = c_2] \leq (1/(2^n - q)) \cdot 0 = 0$. Or in other words, $\Pr[\Delta_1^i = \Delta_1^{i'} \wedge \Delta_2^i = \Delta_2^{i'} \wedge \Delta_3^i = \Delta_3^{i'}] = 0$.

Hence, we can say that for distinct messages $\Pr[\Delta_1^i = \Delta_1^{i'} \wedge \Delta_2^i = \Delta_2^{i'} \wedge \Delta_3^i = \Delta_3^{i'}] \leq 1/(2^n - q)^2$.

Bounding bad cases: BadT_1 . Note that under BadT_1 , we know that there exists at least one pair of indices $i' < i$ such that $T_{\ell i}^i = T_{\ell i'}^{i'}$ and $\Delta_2^i = \Delta_2^{i'}$. Now, as analyzed above we have that for all $i' < i$, the two Δ masks (Δ_1^i, Δ_2^i) are sampled randomly such that the required collisions can only occur with at most probability $2^{n-\min\{n, t-2\}}(1/(2^n - q)^2)$. Since there are total q possible values of i in a session, each having no more than q possible values of i' , we get $\Pr[\text{BadT}_1] \leq \frac{2q^2}{2^{n+\min\{n, t-2\}}}$ for $q \leq 2^{n-1}$.

BadT_2 . Similarly, under BadT_2 , we know that there exists at least one pair of indices $i' < i$ such that $T_{\ell i}^i = T_{\ell i'}^{i'}$, $\Delta_2^i \neq \Delta_2^{i'}$ but $(\text{Tag}_1^i = \text{Tag}_1^{i'}) \vee (\text{Tag}_2^i = \text{Tag}_2^{i'})$. Note that from the same analysis as BadT_1 , the first collision of tweaks here can occur with at most probability of $2^{n-\min\{n, t-2\}}/(2^n - q)$ and further the tags in the ideal world are chosen independently and uniformly at random. Since there are total q possible values of i in a session, each having no more than q possible values of i' , we get $\Pr[\text{BadT}_2] \leq \frac{q^2}{2} \cdot \frac{2}{2^{\min\{n, t-2\}}} \left(\frac{1}{2^n} + \frac{1}{2^n}\right) = \frac{2q^2}{2^{n+\min\{n, t-2\}}}$ for $q \leq 2^{n-1}$.

Now, combining with Exp. 2, we get that $\text{Adv}_{\text{SuperSonic}[(\pi_0, \pi_1)]}^{\text{PRF}}(\mathcal{A}) \leq \frac{4 \cdot q^2}{2^{n+\min\{n, t-2\}}}$ (we drop the condition $q \leq 2^{n-1}$ as for $q > 2^{n-1}$, this bound anyway becomes void) and hence the result of Theorem 2. \square

5.3 DarkSonic: Proof of Theorem 3

Coefficient H Technique. The coefficient H is a simple but powerful proof technique by Patarin [36]. It is often used to prove indistinguishability of a provided construction from an idealized object for an information-theoretic adversary. Coefficient-H based proofs use the concept of “transcripts”. A transcript is defined as a complete record of the interaction of an adversary \mathcal{A} with its oracles in the indistinguishability experiment. For example, if (M_i, T_i) represents the input and output of the i -th query of \mathcal{A} to its oracle and the total number of queries made by \mathcal{A} is q then the corresponding transcript (denoted by τ) is defined as $\tau = \langle (M_1, T_1), \dots, (M_q, T_q) \rangle$. The goal of an adversary \mathcal{A} is to distinguish interactions in the real world $\mathcal{O}_{\text{real}}$ from the ones in ideal world $\mathcal{O}_{\text{ideal}}$.

We denote the distribution of transcripts in the real and the ideal world by Θ_{real} and Θ_{ideal} , respectively. We call a transcript τ *attainable* if the probability of achieving τ in the ideal world is non-zero. Further, w.l.o.g. we also assume that \mathcal{A} does not make any duplicate or prohibited queries. We can now state the fundamental Lemma of coefficient H technique.

Lemma 1 (Fundamental Lemma of the coefficient H Technique [36]). *Consider that the set of attainable transcripts is partitioned into two disjoint sets $\mathcal{T}_{\text{good}}$ and \mathcal{T}_{bad} . Also, assume there exist $\epsilon_1, \epsilon_2 \geq 0$ such that for any transcript $\tau \in \mathcal{T}_{\text{good}}$, we have $\frac{\Pr[\Theta_{\text{real}}=\tau]}{\Pr[\Theta_{\text{ideal}}=\tau]} \geq 1 - \epsilon_1$, and $\Pr[\Theta_{\text{ideal}} \in \mathcal{T}_{\text{bad}}] \leq \epsilon_2$. Then, for all adversaries \mathcal{A} , it holds that*

$$|\Pr[\mathcal{A}^{\mathcal{O}_{\text{real}}}] - \Pr[\mathcal{A}^{\mathcal{O}_{\text{ideal}}}]| \leq \epsilon_1 + \epsilon_2.$$

5.3.1 Proof of Theorem 3

We use the same definition of $\text{Adv}_{\Pi}^{\text{PRF}}(\mathcal{A})$ for a MAC Π as defined in Sec. 5.1.

Replacing F. We first replace $F_{K_2 \oplus T_2}(K_1 \oplus T_1, \cdot)$ with a pair of independent random tweakable permutations $\pi_0 = (\pi_{T_1 \| T_2, 0} \leftarrow \$ \text{Perm}(n))_{T_1 \| T_2 \in \{0,1\}^{\ell+k}}$ and $\pi_1 = (\pi_{T_1 \| T_2, 1} \leftarrow \$ \text{Perm}(n))_{T_1 \| T_2 \in \{0,1\}^{\ell+k}}$ and let $\text{DarkSonic}[(\pi_0, \pi_1)]$ denote the DarkSonic mode that uses π_0, π_1 instead of F, which yields $\text{Adv}_{\text{DarkSonic}[F]}^{\text{PRF}}(\mathcal{A}) \leq \text{Adv}_F^{\text{xrtk-prtfp}}(\mathcal{B}) + \text{Adv}_{\text{DarkSonic}[(\pi_0, \pi_1)]}^{\text{PRF}}(\mathcal{A})$.

Now, the adversary is left with the goal of distinguishing between the games $\text{PRF-real}_{\text{DarkSonic}[(\pi_0, \pi_1)]}$ and $\text{PRF-ideal}_{\text{DarkSonic}[(\pi_0, \pi_1)]}$. For simplicity, we denote these games by “real world” and “ideal world”, respectively. Hence, we want to bound $\text{Adv}_{\text{DarkSonic}[(\pi_0, \pi_1)]}^{\text{PRF}}(\mathcal{A}) = |\Pr[\mathcal{A}^{\text{PRF-real}_{\text{DarkSonic}[(\pi_0, \pi_1)]}}] - \Pr[\mathcal{A}^{\text{PRF-ideal}_{\text{DarkSonic}[(\pi_0, \pi_1)]}}]|$.

Transcripts. Following the coefficient H technique [36], we describe the interactions of \mathcal{A} with its oracles in a *transcript*:

$$\tau = \langle (M^i, \text{Tag}^i)_{i=1}^q \rangle$$

For the i^{th} query to the MAC oracle with input M^i (inc. nonce N^i as M_1^i) and output Tag^i , DarkSonic first adds necessary padding of 10^* in the end of M^i and then internally processes the updated M^i in blocks $P_1^i, \dots, P_{\ell^i-1}^i, P_*^i$ (as defined in the MAC algorithm of DarkSonic, Fig. 8). Here $\ell^i - 1$ represents, the length of M^i in $(n + t + k - 2)$ -bit blocks. DarkSonic also processes internal chaining values defined using underlying permutation calls π_0 s which we denote here by $\Delta^{i,j}$ s. Here (i, j) represents the j^{th} primitive call under the i^{th} message query. We note that the first and the last primitive call in every query require both permutation calls (π_0, π_1) internally and hence generate an extra Δ value each (from their π_1 permutation calls) which we denote by $\Delta^{i,0}$ and Δ^{i,ℓ^i+1} , respectively. Now, the final non-shortened/full $2n$ -bit tag value of Tag^i can be written as $\Delta^{i,\ell^i} \parallel \Delta^{i,\ell^i+1}$.

Additional information. To make the proof analysis simple, we additionally provide the adversary with all the chaining values $\Delta^{i,j}$ s for $0 \leq j \leq \ell^i + 1$ when it has made all its queries and only the final response is pending.

In the real world, all of these Δ variables are internally computed by the MAC oracle that faithfully evaluates DarkSonic. However, in the ideal world, the underlying oracle does not make any computations, and hence $\Delta^{i,j}$ s are not defined. We therefore have to define the sampling of these variables which will be done at the end of the experiment (and thus have no effect on the adversarial queries).

We sample each of the $\Delta^{i,j}$ s with $1 \leq j \leq \ell^i$ uniformly and independently at random, except

1. when such a value is trivially defined due to a “common prefix” with a previous query i.e. when $j \leq \text{llcp}_{n+k+t-2}(M^i, M^{i'})$ for some $i' < i$. To simplify the notations further, we let $\text{llcp}_{n+k+t-2}(i)$ denote $\max_{1 \leq i' < i} \text{llcp}_{n+k+t-2}(M^i, M^{i'})$. Hence, a primitive query (i, j) will be considered as “prefixed-delta” if $j \leq \text{llcp}_{n+k+t-2}(i)$.
2. when such a value is the output of the first fresh primitive call of the query with old/repeating message in the tweak part i.e. when $j = \text{llcp}_{n+k+t-2}(i) + 1$ and $(M^i \oplus M^{i'})[(j-1)(n+t+k-2) + n + 1 \dots (j-1)(n+t+k-2) + n+k+t-2] = 0^{k+t-2}$ for some $i' < i$. In such cases, we sample $\Delta^{i,j}$ s randomly from space $\{0, 1\}^n$ but without replacement (like a random permutation) i.e. if there are x many previous queries $i' < i$ s satisfying the above requirement then $\Delta^{i,j}$ s are sampled randomly with probability $1/(2^n - x)$ from the space $\{0, 1\}^n$ with excluding all x many $\Delta^{i',j}$ s.

Further, we sample $\Delta^{i,0}$ (and Δ^{i,ℓ^i+1}) identically to $\Delta^{i,1}$ (and Δ^{i,ℓ^i}) but with redefining the sampling excluded set using previous x many $\Delta^{i',0}$ (and $\Delta^{i',\ell^{i'}+1}$) values, respectively. Clearly, this give away of additional information can only help the adversary by increasing its advantage and hence can be considered here for upper bounding the targeted (above mentioned) adversarial advantage.

Extended transcripts. With the defined additional information to the adversary, we can now re-define the extended transcripts as

$$\tau = \left\langle \left((P_j^i)_{j=1}^{\ell^i}, (\Delta^{i,j})_{j=0}^{\ell^i+1} \right)_{i=1}^q \right\rangle$$

where $P_{\ell^i}^i = P_*^i$.

Coefficient-H. Let us represent the distribution of the transcript in the real world and the ideal world by Θ_{re} and Θ_{id} , respectively.

The proof relies on the fundamental lemma of the coefficient H technique as defined in Lemma 1 above. We say an attainable transcript τ is bad if one of the following conditions occurs:

BadT₁ a.k.a. “Input Collision”: There exists $(i', j') < (i, j)$ with $(j' > \text{llcp}_{n+k+t-2}(i')) \wedge (j > \text{llcp}_{n+k+t-2}(i))$, $1 \leq j \leq \ell^i$, $1 \leq j' \leq \ell^{i'}$ and $\max\{j, j'\} \neq 1$ such that, the $(i, j)^{th}$ primitive call has tweak-input collision with the $(i', j')^{th}$ primitive call, i.e. for $z = \min\{n, t-2\}$

1. with $(j \notin \{1, \ell^i\} \wedge j' \notin \{1, \ell^{i'}\}) \vee (j = \ell^i \wedge j' = \ell^{i'})$, $P_j^i \oplus P_{j'}^{i'} = (\Delta^{i,j-2} \oplus \Delta^{i',j'-2}) \parallel (\Delta^{i,j-1} \oplus \Delta^{i',j'-1})[1 \dots z] \parallel 0^{k+t-2-z}$
2. or with $j = 1 \wedge j' \notin \{1, \ell^{i'}\}$, $P_j^i \oplus P_{j'}^{i'} = \Delta^{i',j'-2} \parallel \Delta^{i',j'-1}[1 \dots z] \parallel 0^{k+t-2-z}$
3. or with $j \notin \{1, \ell^i\} \wedge j' = 1$, $P_j^i \oplus P_{j'}^{i'} = \Delta^{i,j-2} \parallel \Delta^{i,j-1}[1 \dots z] \parallel 0^{k+t-2-z}$

BadT₂ a.k.a. “Output Collision”: There exists $(i', j') < (i, j)$ with $(j' > \text{lcp}_{n+k+t-2}(i')) \wedge (j > \text{lcp}_{n+k+t-2}(i))$, $1 \leq j \leq \ell^i$, $1 \leq j' \leq \ell^{i'}$ and $\max\{j, j'\} \neq 1$ such that, the $(i, j)^{\text{th}}$ primitive call has tweak-output collisions with the $(i', j')^{\text{th}}$ primitive call given that the inputs to these calls are distinct, i.e. for $z = \min\{n, t-2\}$

1. with $(j \notin \{1, \ell^i\} \wedge j' \neq 1) \vee (j = \ell^i \wedge j' \notin \{1, \ell^{i'}\})$, $(P_j^i \oplus P_{j'}^{i'})[n+1 \dots n+k+t-2] = (\Delta^{i,j-1} \oplus \Delta^{i',j'-1})[1 \dots z] \parallel 0^{k+t-2-z}$, $(P_j^i \oplus P_{j'}^{i'})[1 \dots n] \neq \Delta^{i,j-2} \oplus \Delta^{i',j'-2}$ and $\Delta^{i,j} = \Delta^{i',j'}$
3. or with $j \notin \{1, \ell^i\} \wedge j' = 1$, $(P_j^i \oplus P_{j'}^{i'})[n+1 \dots n+k+t-2] = \Delta^{i,j-1}[1 \dots z] \parallel 0^{k+t-2-z}$, $(P_j^i \oplus P_{j'}^{i'})[1 \dots n] \neq \Delta^{i,j-2}$ and $\Delta^{i,j} = \Delta^{i',j'}$
3. or with $j = 1 \wedge j' \notin \{1, \ell^{i'}\}$, $(P_j^i \oplus P_{j'}^{i'})[n+1 \dots n+k+t-2] = \Delta^{i',j'-1}[1 \dots z] \parallel 0^{k+t-2-z}$, $(P_j^i \oplus P_{j'}^{i'})[1 \dots n] \neq \Delta^{i',j'-2}$ and $\Delta^{i,j} = \Delta^{i',j'}$
4. or with $j = \ell^i \wedge j' = \ell^{i'}$, $(P_j^i \oplus P_{j'}^{i'})[n+1 \dots n+k+t-2] = (\Delta^{i,j-1} \oplus \Delta^{i',j'-1})[1 \dots z] \parallel 0^{k+t-2-z}$, $(P_j^i \oplus P_{j'}^{i'})[1 \dots n] \neq \Delta^{i,j-2} \oplus \Delta^{i',j'-2}$ and $(\Delta^{i,j} = \Delta^{i',j'} \vee \Delta^{i,j+1} = \Delta^{i',j'+1})$

We note that these collisions cannot occur in the real world where the tags are generated using permutation but they can still occur in the ideal world. We also note that for the missing possible cases of (j, j') here, the targeted tweak-input (or tweak-output, respectively) pairs of any primitive query pairs are always distinct (due to tweak T₁'s domain separating last two bits or both primitive queries being the first fresh primitive queries of their corresponding messages) and thus are trivially excluded from the bad cases BadT₁ (or BadT₂, respectively). We denote by \mathcal{T}_{bad} , the set of “bad” transcripts that is defined as the subset of attainable transcripts for which the transcript predicate $\text{BadT}(\tau) = (\text{BadT}_1(\tau) \vee \text{BadT}_2(\tau)) = 1$. We denote by $\mathcal{T}_{\text{good}}$, the set of attainable transcripts which are not in the set \mathcal{T}_{bad} .

Lemma 2. *For \mathcal{T}_{bad} above, we have*

$$\Pr[\Theta_{id} \in \mathcal{T}_{\text{bad}}] \leq \frac{6(\sigma - q)^2}{2^{n+\min\{n,t-2\}}} + \frac{2q(\mu - 1)}{2^{\min\{n,t-2\}}}.$$

Lemma 3. *Let $\tau \in \mathcal{T}_{\text{good}}$ i.e. τ is a good transcript. Then $\frac{\Pr[\Theta_{re}=\tau]}{\Pr[\Theta_{id}=\tau]} \geq 1$.*

With the well-defined bad events, both lemmas can be proved using standard probability analysis. We defer the proof of Lemma 2 and 3 to Appendix A.

Combining the results of Lemma 2 and 3 (taking $\epsilon_1 = 0$) into Lemma 1, we obtain the upper bound $\text{Adv}_{\text{DarkSonic}^{\text{PRF}}[(\pi_0, \pi_1)]}(\mathcal{A}) \leq \frac{6(\sigma - q)^2}{2^{n+\min\{n,t-2\}}} + \frac{2q(\mu - 1)}{2^{\min\{n,t-2\}}}$ and hence the result of Theorem 3. \square

6 SonicAE: SuperSonic based deterministic AE

Following PMAC2x based SIVx [32] and ZMAC based ZAE [27], we also use SuperSonic to construct a stateless or deterministic authenticated encryption (DAE) [41] scheme. A DAE is an AE scheme whose security does not rely on additional inputs such as random IVs or nonces. We refer the reader to [41] for more details on DAE syntax and security (known as *dae*). We note that a DAE scheme with *dae* security can also take a nonce input as part of the associated data (AD) and hence by definition provides security against nonce-misuse. Such schemes and achieved security are known as misuse resistant AEs (MRAEs) and *mrae*, respectively.

As a result, we provide SonicAE, an SIV [41]-like composition of SuperSonic MAC and GCTR₂'-3 encryption mode as shown in Fig. 6. Here GCTR₂'-3 is defined as GCTR₂-3 [2] mode but with two changes; 1. tweak domain separated from SuperSonic by fixing the last two tweak-bits to 10 and 2. nonce input replaced by the second half of $2n$ -bit *Tag* value.

SonicAE is a *dae*-secure scheme that provides beyond birthday security of $\min\{n, (n+t)/2\}$ bits (see Theorem 4) and processes on average $2n(n+t+k)/(3n+t+k)$ bits per FC call. We note that for the least settings of t and k i.e. $t=0$ and $k=n$, we get a processing rate of n bits per FC call which is same as the existing FC-based AE schemes [6] PAEF, SAEF and RPAEF, however unlike them SonicAE provides stronger security guarantee of *dae*. On the other hand, for higher settings when one increases t and/or k , the processing rate of SonicAE moves towards $2n$ bits per FC call which is really high and better when compared with the SotA AE schemes ZAE and Deoxys-II [29].

For concrete improvements, we provide in Fig. 7 and Table 3, a comparison of SonicAE with other SotA AEs - ZAE, Deoxys-I [29], Deoxys-II, OCB3 [30], SIV [41] and CCM [43] where all FCs and TBCs are instantiated with ForkSkinny and SKINNY, respectively. For the same, we set AD length to 0 to obtain a conservative prediction of speedup, seeing as increasing the AD length improves the performance of SonicAE.

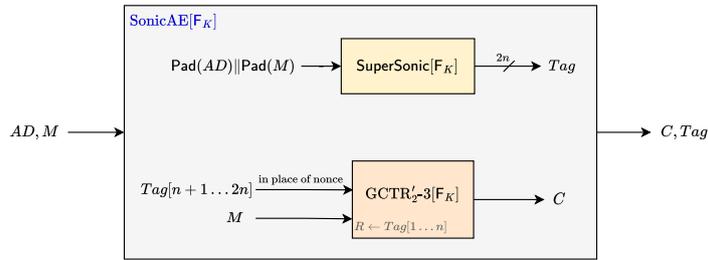


Figure 6: The SonicAE AEAD mode. The picture illustrates the processing of arbitrary length associated data AD and message M using the MAC mode SuperSonic and encryption mode GCTR₂'-3 with key K .

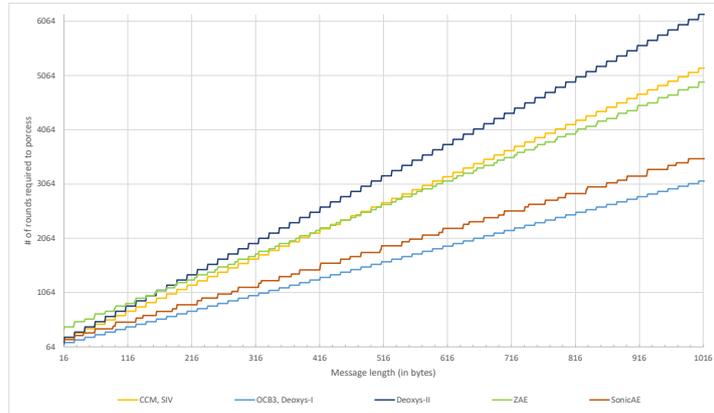


Figure 7: Efficiency comparison of SonicAE (defined as the SIV composition of SuperSonic and GCTR₂'-3) with other SotA AEs under setting $n = k = 128$ bits. All TBCs and FCs are instantiated with SKINNY and ForkSkinny, respectively. The plot shows the number of rounds that are required to process a message of corresponding length.

Clearly, from Fig. 7 and Table 3 we can infer that SonicAE performs significantly better than the SotA AEs that provide some level of nonce-misuse (*mrae*) security. In particular, SonicAE[ForkSkinny] gains a speed-up of at least factor 1.41x and 1.32x against existing MRAE schemes ZAE[SKINNY] and Deoxys-II[SKINNY] under $k = 128$, $t = n \in \{64, 128\}$ for small queries of size 95B and large queries up to 64KB, respectively.

Table 3: Efficiency comparison of SonicAE (instantiated with ForkSkinny) against the SotA AE modes (instantiated with SKINNY) under settings $k = 128$ bits, $n \in \{64, 128\}$ bits and $t \in \{0, n\}$ depending on the need of the compared AE mode. The entries in this table show the ratio of number of rounds required by the corresponding SotA AE mode and by SonicAE (under same values of n, t and k) to process a message of the given maximum length. Here **green** and **red** cells represent settings where SonicAE is better and worse than the compared mode, respectively.

AE →	ZAE		Deoxys-II		OCB3, Deoxys-I		CCM, SIV	
Block size n (in bits) →	64	128	64	128	64	128	64	128
IT nAE Security (in bits) →	64	128	64	128	32, 64	64, 128	32	64
IT MRAE Security (in bits) →	64	128	<64	<128	NIL	NIL	NIL, 32	NIL, 64
Max. message length	SonicAE speed-up/slow-down (in multiplicative factors) against the mode							
16B	2.41x	1.94x	1.20x	0.73x	0.72x	0.48x	1.30x	0.81x
95B	1.56x	1.73x	1.56x	1.41x	0.81x	0.76x	1.46x	1.26x
4KB	1.33x	1.35x	1.74x	1.77x	0.87x	0.88x	1.57x	1.47x
64KB	1.32x	1.34x	1.75x	1.78x	0.87x	0.89x	1.57x	1.48x

Security of SonicAE. We state the formal claim about the dae security of SonicAE in Theorem 4 and defer its proof to Appendix B.

Theorem 4. *Let F be a tweakable expanding primitive with $\mathcal{T} = \{0, 1\}^t$. Then for any adversary \mathcal{A} who makes at most q queries to the SonicAE such that the total number of primitive calls induced by all the queries is at most σ , we have $\text{Adv}_{\text{SonicAE}[F]}^{\text{dae}}(\mathcal{A})$*

$$\leq \text{Adv}_F^{\text{xrtk-prtfp}}(\mathcal{B}) + \frac{3q(\sigma + q)}{2^{n+\min\{n,t-2\}}} + \frac{q}{2^{2n}}$$

for some adversary \mathcal{B} , making at most σ queries, and running in time given by the running time of \mathcal{A} plus $\gamma \cdot \sigma$ where γ is the runtime of an F call in the model of computation.

7 Conclusion

We proposed Sonikku family consisting of two sequential MACs BabySonic and DarkSonic, and one fully parallelizable MAC SuperSonic. BabySonic outperforms ZMAC in applications where parallelization is not available and small footprint is desired. It is also *overall the best MAC choice for small queries* in terms of both security and efficiency as it does not have pre- or post-processing overhead. It also comes with “*optimal*” state size. BabySonic gains a speed-up of at least **2.11x** (and up to **4.36x**) against ZMAC with $n = t = k = 128$ for queries of size 95B or smaller.

When compared against BabySonic, DarkSonic improves its security level and performance by a reasonably large margin with the use of only one primitive leg for most of the processing, and adding one pre-processing call for the mandatory nonce. Even when the nonce is repeated, DarkSonic provides security that only degrades logarithmically with the number of nonce repetitions.

Finally, SuperSonic comes with security that is message length independent and has no nonce requirement. The design is fully parallelizable and offers comparable performance to DarkSonic when run sequentially at the cost of a relatively large state size. DarkSonic and SuperSonic gain a speed-up of at least **1.93x** and **1.48x** against ZMAC with $k = 128$, $t = n \in \{64, 128\}$ for small queries of size 95B (or smaller) and large queries up to 64KB, respectively.

Acknowledgments

This work was supported by CyberSecurity Research Flanders with reference number VR20192203. This work was supported in part by the Research Council KU Leuven C1 on

Security and Privacy for Cyber-Physical Systems and the Internet of Things with contract number C16/15/058 and by the Flemish Government through FWO Project G.0835.16 A security Architecture for IoT.

References

- [1] BRICK Radio Remote Control. 2018. <https://www.controldevices.group/PDFS/TER/TER%20Brick%20Radio%20Remote%20Control%20Data%20Sheet.pdf>.
- [2] Elena Andreeva, Amit Singh Bhati, Bart Preneel, and Damian Vizár. 1, 2, 3, Fork: Counter Mode Variants based on a Generalized Forkcipher. *IACR Trans. Symmetric Cryptol.*, 2021(3):1–35, 2021.
- [3] Elena Andreeva, Amit Singh Bhati, and Damian Vizár. Nonce-misuse security of the SAEF authenticated encryption mode. In *Selected Areas in Cryptography: 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21–23, 2020, Revised Selected Papers 27*, pages 512–534. Springer, 2021.
- [4] Elena Andreeva, Benoit Cogliati, Virginie Lallemand, Marine Minier, Antoon Purnal, and Arnab Roy. Masked Iterate-Fork-Iterate: A new Design Paradigm for Tweakable Expanding Pseudorandom Function. In *22nd International Conference on Applied Cryptography and Network Security*, 2024.
- [5] Elena Andreeva, Arne Deprez, Jowan Pittevels, Arnab Roy, Amit Singh Bhati, and Damian Vizár. New results and insights on forkcae. In *NIST LWC workshop*, 2020.
- [6] Elena Andreeva, Virginie Lallemand, Antoon Purnal, Reza Reyhanitabar, Arnab Roy, and Damian Vizár. Forkcipher: a New Primitive for Authenticated Encryption of Very Short Messages. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 153–182. Springer, 2019.
- [7] Elena Andreeva and Andreas Wenzinger. A forkcipher-based pseudo-random number generator. In *International Conference on Applied Cryptography and Network Security*, pages 3–31. Springer, 2023.
- [8] 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.
- [9] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental Cryptography: The Case of Hashing and Signing. In Yvo G. Desmedt, editor, *Advances in Cryptology — CRYPTO '94*, pages 216–233, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.
- [10] Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 491–506. Springer, 2003.
- [11] Francesco Berti, François-Xavier Standaert, and Itamar Levi. Authenticity in the presence of leakage using a forkcipher. *Cryptology ePrint Archive*, Paper 2024/1325, 2024.
- [12] Karthikeyan Bhargavan and Gaëtan Leurent. On the Practical (In-)Security of 64-Bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16*, page 456–467, New York, NY, USA, 2016. Association for Computing Machinery.

- [13] Amit Singh Bhati, Elena Andreeva, and Damian Vizár. OAE-RUP: a strong online AEAD security notion and its application to SAEF. In *International Conference on Security and Cryptography for Networks*, pages 117–139. Springer, 2024.
- [14] Amit Singh Bhati, Antonín Dufka, Elena Andreeva, Arnab Roy, and Bart Preneel. Skye: An Expanding PRF based Fast KDF and its Applications. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, pages 1082–1098, 2024.
- [15] Amit Singh Bhati, Erik Pohle, Aysajan Abidin, Elena Andreeva, and Bart Preneel. Let’s Go Eevee! A Friendly and Suitable Family of AEAD Modes for IoT-to-Cloud Secure Computation. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 2546–2560, 2023.
- [16] Amit Singh Bhati, Michiel Verbauwhede, and Elena Andreeva. Breaking, Repairing and Enhancing XCBv2 into the Tweakable Enciphering Mode GEM. *Cryptology ePrint Archive*, 2024. <https://eprint.iacr.org/2024/1554>.
- [17] John Black and Phillip Rogaway. A Block-Cipher Mode of Operation for Parallelizable Message Authentication. In *EUROCRYPT 2002*, pages 384–397, 2002.
- [18] Benoît Cogliati, Jooyoung Lee, and Yannick Seurin. New Constructions of MACs from (Tweakable) Block Ciphers. *IACR Transactions on Symmetric Cryptology*, 2017(2):27–58, 2017.
- [19] Nilanjan Datta, Avijit Dutta, Eik List, and Sougata Mandal. FEDT: Forkcipher-based leakage-resilient beyond-birthday-secure AE. *IACR Communications in Cryptology*, 1(2), 2024.
- [20] Nilanjan Datta, Avijit Dutta, and Cuauhtemoc Mancillas-López. LightMAC: Fork it and make it faster. *Advances in Mathematics of Communications*, 18(5):1406–1441, 2024.
- [21] Nilanjan Datta, Avijit Dutta, Mridul Nandi, Goutam Paul, and Liting Zhang. Single key variant of PMAC_Plus. *IACR Transactions on Symmetric Cryptology*, pages 268–305, 2017.
- [22] Xiaoyang Dong, Lingyue Qin, Siwei Sun, and Xiaoyun Wang. Key guessing strategies for linear key-schedule algorithms in rectangle attacks. In *Advances in Cryptology–EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30–June 3, 2022, Proceedings, Part III*, pages 3–33. Springer, 2022.
- [23] Avijit Dutta, Jian Guo, and Eik List. Forking Sums of Permutations for Optimally Secure and Highly Efficient PRFs. *Cryptology ePrint Archive*, 2022.
- [24] Peter Gaži, Krzysztof Pietrzak, and Michal Rybár. The exact prf-security of nmac and hmac. *Cryptology ePrint Archive*, Paper 2014/578, 2014. <https://eprint.iacr.org/2014/578>.
- [25] Aldo Gunsing and Bart Mennink. The Summation-Truncation Hybrid: Reusing Discarded Bits for Free. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, pages 187–217, Cham, 2020. Springer International Publishing.
- [26] Tetsu Iwata and Kaoru Kurosawa. OMAC: One-Key CBC MAC. In Thomas Johansson, editor, *Fast Software Encryption*, pages 129–153, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

- [27] Tetsu Iwata, Kazuhiko Minematsu, Thomas Peyrin, and Yannick Seurin. ZMAC: a fast tweakable block cipher mode for highly secure message authentication. In *Annual international cryptology conference*, pages 34–65. Springer, 2017.
- [28] 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.
- [29] Jérémy Jean, Ivica Nikolić, Thomas Peyrin, and Yannick Seurin. Deoxys v1.41, 2016. <https://competitions.cr.yp.to/round3/deoxysv141.pdf>.
- [30] Ted Krovetz and Phillip Rogaway. The Software Performance of Authenticated-Encryption Modes. In Antoine Joux, editor, *Fast Software Encryption*, pages 306–327, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- [31] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable Block Ciphers. In *CRYPTO 2002*, pages 31–46, 2002.
- [32] Eik List and Mridul Nandi. Revisiting full-PRF-secure PMAC and using it for beyond-birthday authenticated encryption. In *Cryptographers’ Track at the RSA Conference*, pages 258–274. Springer, 2017.
- [33] Guozhen Liu, Mohona Ghosh, and Ling Song. Security analysis of SKINNY under related-tweakey settings. *Cryptology ePrint Archive*, 2016.
- [34] Atul Luykx, Bart Preneel, Elmar Tischhauser, and Kan Yasuda. A MAC mode for lightweight block ciphers. In *International Conference on Fast Software Encryption*, pages 43–59. Springer, 2016.
- [35] Yusuke Naito. Full PRF-secure message authentication code based on tweakable block cipher. In *International Conference on Provable Security*, pages 167–182. Springer, 2015.
- [36] Jacques Patarin. *The “Coefficients H” Technique*, page 328–345. Springer-Verlag, Berlin, Heidelberg, 2009.
- [37] Trevor Perrin. The Noise protocol framework. 2016. noiseprotocol.org.
- [38] Antoon Purnal, Elena Andreeva, Arnab Roy, and Damian Vizár. What the Fork: Implementation Aspects of a Forkcipher. In *NIST Lightweight Cryptography Workshop 2019*, 2019.
- [39] Lingyue Qin, Xiaoyang Dong, Xiaoyun Wang, Keting Jia, and Yunwen Liu. Automated Search Oriented to Key Recovery on Ciphers with Linear Key Schedule. *IACR Transactions on Symmetric Cryptology*, pages 249–291, 2021.
- [40] Phillip Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In Pil Joong Lee, editor, *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 16–31. Springer, 2004.
- [41] 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.
- [42] Soliton. Teleoperation of Remote Machinery. <https://www.solitonsystems.com/low-latency-video/remote-operation/remote-operation-of-machinery>.
- [43] D. Whiting, R. Housley, and N. Ferguson. Counter with CBC-MAC (CCM). IETF RFC 3610 (Informational), September 2003.

- [44] Kan Yasuda. The sum of CBC MACs is a secure PRF. In *Cryptographers' Track at the RSA Conference*, pages 366–381. Springer, 2010.
- [45] Kan Yasuda. A new variant of PMAC: beyond the birthday bound. In *Annual Cryptology Conference*, pages 596–609. Springer, 2011.
- [46] YOURCONTROL. The Proxo Wireless Remote Control System. <https://your-control.com/proxo/>.
- [47] 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. *Designs, Codes and Cryptography*, 88(6):1103–1126, 2020.

A Omitted Lemma Proofs

A.1 Proof of Lemma 2

Proof. **BadT₁.** For any transcript in \mathcal{T}_{bad} with **BadT₁** set to 1, we know that there exists $(i', j') < (i, j)$ with $(j' > \text{lcp}_{n+k+t-2}(i')) \wedge (j > \text{lcp}_{n+k+t-2}(i))$, $1 \leq j \leq \ell^i$, $1 \leq j' \leq \ell^{i'}$ and $\max\{j, j'\} \neq 1$ such that, one of the three statements of **BadT₁** (as defined above) holds.

Note that under any of these statements, we have the following cases for the mentioned Δ_s (being $\Delta^{i,j-1}$, $\Delta^{i',j'-1}$, $\Delta^{i,j-2}$ or $\Delta^{i',j'-2}$).

- I. When $j = j' = \text{lcp}_{n+k+t-2}(M^i, M^{i'}) + 1$: $\Delta^{i,j-1} = \Delta^{i',j'-1}$ and $\Delta^{i,j-2} = \Delta^{i',j'-2}$. However, the message parts $P_j^i \neq P_{j'}^{i'}$ and therefore, any of the targeted **BadT₁** collisions can occur here with probability 0.
- II. When $j = j' = \text{lcp}_{n+k+t-2}(M^i, M^{i'}) + 2 > 2$: $\Delta^{i,j-2} = \Delta^{i',j'-2}$. However, $\Delta^{i,j-1}$ and $\Delta^{i',j'-1}$ (if defined) are fresh and chosen uniformly at random from a subspace of $\{0, 1\}^n$ with probability at most $(1/(2^n - q))$ each. This implies that any of the targeted **BadT₁** collisions can occur here with probability $\leq 2^{n-z}/(2^n - q)$.
- III. Otherwise: all mentioned Δ_s here are relatively fresh which means in the ideal world, each one of these Δ_s even when fixing the rest three, is chosen uniformly at random from a subspace of $\{0, 1\}^n$ with probability at most $(1/(2^n - q))$ each (“at most” because for message queries, the very first fresh Δ_s are chosen using a random permutation or a random permutation pair without replacement). Hence, any of the targeted **BadT₁** collisions can occur here with probability $\leq 2^{n-z}/(2^n - q)^2$.

Now, since there are total of q many message queries containing $\sigma \geq 2q$ many total primitive calls, we have at most $\binom{\sigma-q}{2}$ many (among all except last primitive calls) and $\binom{q}{2}$ many (among the last primitive calls) possible pairs of $(i', j') < (i, j)$ satisfying all the three statements of **BadT₁**. Out of these, $\binom{q}{2}$ and at most $\binom{q}{1} \binom{\mu-1}{1}$ pairs satisfy the above conditions I and II, respectively where μ denotes the maximum number of times a nonce N can repeat over q queries. With this, we get under $q \leq 2^{n-1}$, $\Pr[\text{BadT}_1(\Theta_{id}) = 1] \leq \frac{\binom{\sigma-q}{2} + \binom{q}{2} - q(\mu-1)}{2^{n+z-2}} + \frac{q(\mu-1)}{2^{z-1}} \leq \frac{(\sigma-q)^2}{2^{n+z-1}} + \frac{q(\mu-1)}{2^{z-1}}$.

BadT₂. Similarly, for any transcript in \mathcal{T}_{bad} with **BadT₂** set to 1, we know that there exists $(i', j') < (i, j)$ with $(j' > \text{lcp}_{n+k+t-2}(i')) \wedge (j > \text{lcp}_{n+k+t-2}(i))$, $1 \leq j \leq \ell^i$, $1 \leq j' \leq \ell^{i'}$ and $\max\{j, j'\} \neq 1$ such that, one of the four statements of **BadT₂** (as defined above) holds.

Note that under any of these statements, we have the following cases for the mentioned Δ_s (being $\Delta^{i,j-1}$, $\Delta^{i',j'-1}$, $\Delta^{i,j}$, $\Delta^{i',j'} \Delta^{i,j+1}$ or $\Delta^{i',j'+1}$).

- I. When $j = j' = \text{lcp}_{n+k+t-2}(M^i, M^{i'}) + 1$: Here distinct inputs with same tweak imply that $\Delta^{i,j} \neq \Delta^{i',j'}$ (and additionally when $j = \ell^i = \ell^{i'}$, $\Delta^{i,j+1} \neq \Delta^{i',j'+1}$) and therefore, any of the targeted BadT_2 collisions can occur here with probability 0.
- II. Otherwise: all mentioned Δ s here are relatively fresh which means in the ideal world, each one of these Δ s even when fixing the rest five, is chosen uniformly at random from a subspace of $\{0, 1\}^n$ with probability at most $(1/(2^n - q))$ each. Hence, any of the targeted BadT_2 collisions can occur with probability $\leq 2^{n-z} \cdot (2/(2^n - q)^2)$.

Now, since there are total of q many message queries containing σ many total primitive calls, we have at most $(\frac{\sigma-q}{2})$ many (among all except last primitive calls) and $(\frac{q}{2})$ many (among the last primitive calls) possible pairs of $(i', j') < (i, j)$ satisfying the four statements. Out of these, $(\frac{q}{2})$ pairs satisfy the above conditions I. Hence, we get under $q \leq 2^{n-1}$, $\Pr[\text{BadT}_2(\Theta_{id}) = 1] \leq \frac{(\frac{\sigma-q}{2}) + (\frac{q}{2}) - (\frac{q}{2})}{2^{n+z-3}} \leq \frac{2(\sigma-q)^2}{2^{n+z-1}}$.

Thus, we obtain by the union bound that $\Pr[\Theta_{id} \in \mathcal{T}_{\text{bad}}] \leq \frac{6(\sigma-q)^2}{2^{n+\min\{n, t-2\}}} + \frac{2q(\mu-1)}{2^{\min\{n, t-2\}}}$. \square

A.2 Proof of Lemma 3

Proof. Note that a good transcript has the following property that for each $(i', j') < (i, j)$ if the permutation pairs have same tweaks (i.e. the $\mathcal{T}_1 \parallel \mathcal{T}_2$ part) then these pairs will always have different inputs and different outputs.

The probability to obtain a good transcript τ in the real and the ideal worlds can now be computed. Let x denote the number of total permutation queries (π_0 and π_1 are considered two different queries) that return prefixed-deltas (i.e. queries that output some repeated old Δ s within τ due to sharing a common prefix in their corresponding message with some previously queried message) over all $\sigma + 2q$ permutation calls. Let $y_{\mathcal{T}'}$ and $y'_{\mathcal{T}'}$ denote the number of permutation queries (i, j, b) s that share a same tweak $\mathcal{T}_1 \parallel \mathcal{T}_2 \parallel b$ as \mathcal{T}' (for some $\mathcal{T}' \in \{0, 1\}^{t+k+1}$) over all $\sigma + 2q$ permutation calls that return no prefixed deltas and over just the first fresh primitive (can be a permutation or a pair of them) calls, respectively. Here $b \in \{0, 1\}$ is a bit defining the index of permutation π_b . Also, let us denote the set of all distinct tweaks used in a session of $\sigma + 2q$ permutation queries and just the first fresh primitive queries as $\mathcal{T}_{\text{total}}$ and $\mathcal{T}_{\text{first}}$, respectively. Clearly, with this, we can say $\sum_{\mathcal{T}' \in \mathcal{T}_{\text{total}}} y_{\mathcal{T}'} = \sigma + 2q - x$ and $y_{\mathcal{T}'} \geq y'_{\mathcal{T}'}$.

Since in the ideal world, all these non-prefixed Δ s are sampled uniformly and independently at random except when they are the outputs of the first fresh primitive query of their messages and are sampled using a random permutation (or a pair of these), we get for $g = \sum_{\mathcal{T}' \in \mathcal{T}_{\text{first}}} y'_{\mathcal{T}'}$, $\Pr[\Theta_{id} = \tau] = (1^x \cdot (1/2^n)^{(\sigma+2q-x)-g} \cdot (\prod_{\mathcal{T}' \in \mathcal{T}_{\text{first}}} 1/(2^n)^{y'_{\mathcal{T}'}}))$. On the other hand, in the real world these entities are computed using random tweaked permutations for similar queries which gives $\Pr[\Theta_{re} = \tau] = (1^x \cdot (\prod_{\mathcal{T}' \in \mathcal{T}_{\text{total}}} 1/(2^n)^{y_{\mathcal{T}'}}))$ and consequently

$$\begin{aligned} \frac{\Pr[\Theta_{re} = \tau]}{\Pr[\Theta_{id} = \tau]} &= \left(\frac{2^{n((\sigma+2q-x)-g)} \cdot \prod_{\mathcal{T}' \in \mathcal{T}_{\text{first}}} (2^n)^{y'_{\mathcal{T}'}}}{\prod_{\mathcal{T}' \in \mathcal{T}_{\text{total}}} (2^n)^{y_{\mathcal{T}'}}} \right) \\ &= \left(\frac{\prod_{\mathcal{T}' \in \mathcal{T}_{\text{total}}} (2^n)^{y_{\mathcal{T}'}} \cdot \prod_{\mathcal{T}' \in \mathcal{T}_{\text{first}}} (2^n)^{y'_{\mathcal{T}'}}}{\prod_{\mathcal{T}' \in \mathcal{T}_{\text{first}}} (2^n)^{y'_{\mathcal{T}'}} \cdot \prod_{\mathcal{T}' \in \mathcal{T}_{\text{total}}} (2^n)^{y_{\mathcal{T}'}}} \right) \\ &= \left(\frac{\prod_{\mathcal{T}' \in \mathcal{T}_{\text{total}}} \frac{(2^n)^{y_{\mathcal{T}'}}}{(2^n)^{y_{\mathcal{T}'}}} \cdot \prod_{\mathcal{T}' \in \mathcal{T}_{\text{first}}} \frac{(2^n)^{y'_{\mathcal{T}'}}}{(2^n)^{y'_{\mathcal{T}'}}} \right) \geq 1. \end{aligned}$$

We note that the last inequality holds here because $y_{\mathcal{T}'} \geq y'_{\mathcal{T}'}$ and $\mathcal{T}_{\text{first}} \subseteq \mathcal{T}_{\text{total}}$. \square

B Proof of Theorem 4

Proof of Theorem 4. We provide the proof for the information-theoretic (IT) security of $\text{SonicAE}[(\pi_0, \pi_1)]$ where (π_0, π_1) is a pair of independent random tweakable permutations $\pi_0 = (\pi_{\tau_1 \parallel \tau_2, 0} \leftarrow \$ \text{Perm}(n))_{\tau_1 \parallel \tau_2 \in \{0,1\}^{t+k}}$ and $\pi_1 = (\pi_{\tau_1 \parallel \tau_2, 1} \leftarrow \$ \text{Perm}(n))_{\tau_1 \parallel \tau_2 \in \{0,1\}^{t+k}}$ replacing F in SonicAE (then from there, the proof for the computational counterpart is standard and straightforward). The proof of IT security of SonicAE follows as a corollary from Theorem 2 and the following two results derived in order from [41, Theorem 2] and [2, GCTR₂₋₃ proof and App. B.2], respectively: there exists an adversary \mathcal{A}' attacking SuperSonic and an adversary \mathcal{A}'' attacking GCTR'_{2-3} , both making at most q queries such that the total number of primitive calls induced is at most σ and we have

$$\text{Adv}_{\text{SonicAE}[(\pi_0, \pi_1)]}^{\text{dae}}(\mathcal{A}) \leq \text{Adv}_{\text{SuperSonic}[(\pi_0, \pi_1)]}^{\text{prf}}(\mathcal{A}') + \frac{q}{2^{2n}} + \text{Adv}_{\text{GCTR}'_{2-3}[(\pi_0, \pi_1)]}^{\text{ive}}(\mathcal{A}''), \quad (3)$$

$$\begin{aligned} \text{Adv}_{\text{GCTR}'_{2-3}[(\pi_0, \pi_1)]}^{\text{ive}}(\mathcal{A}'') &\leq \frac{2(2\sigma - q)q}{2^{n+\min\{n, t-2\}+1}} + \Pr[\mathcal{V} \text{ for GCTR}'_{2-3} \mid r = \min\{n, t-2\}] \\ &\leq \frac{(2\sigma - q)q}{2^{n+\min\{n, t-2\}}} + \Pr[\mathcal{V} \text{ for GCTR}_{2-3} \mid x = q \wedge r = \min\{n, t-2\}] \cdot \max_{i \neq i' \leq q} \{\Pr[N^i = N^{i'}]\} \\ &\leq \frac{(2\sigma - q)q}{2^{n+\min\{n, t-2\}}} + \frac{(2\sigma - q)q}{2^{\min\{n, t-2\}+1}} \cdot \frac{1}{2^n} = \frac{6q\sigma - 3q^2}{2^{n+\min\{n, t-2\}+1}} \end{aligned} \quad (4)$$

where ive is the IV-based encryption security notion and \mathcal{V} is the event of cross-query input-tweak pair collisions. We refer the reader to [2] for more details on these used notations. Here $\max_{i \neq i' \leq q} \{\Pr[N^i = N^{i'}]\}$ denotes the maximum probability of a nonce repetition over q queries which is equal to $1/2^n$ here as nonce inputs in GCTR'_{2-3} are defined as the second halves of uniform random $2n$ -bit tags. Now, simply combining Exp. 3, 4 and the result from Theorem 2 gives us the corresponding IT security claim of Theorem 4. \square

C Sonikku: Pseudocodes

<pre> 1: function Pad(x, y, M) // y ≤ x 2: l'pad ← 01; res ← M %x 3: if res ≠ y then 4: l'pad ← 11 5: if res < y then 6: M ← M 10^{y-1-res} 7: else 8: M ← M 10^{x+y-1-res} 9: end if 10: end if 11: return M, l'pad 12: end function 13: 14: function MAC(K, a, τ, M) // k ≤ n + a 15: K₁, K₂ ← Derive(K) 16: M, l'pad ← Pad(n + t + k - 2, 0, M) 17: P₁, ..., P_{ℓ-1}, P* ← $\frac{n+t+k-2}{M}$ 18: (Δ₁, Δ₂) ← (K₁, K₂) </pre>	<pre> 19: for i ← 1 to ℓ - 1 do 20: M_{3i-2} ← P_i[1...n] 21: M_{3i-1} ← P_i[n+1...n+t-2] 22: M_{3i} ← P_i[n+t-1...n+t+k-2] 23: T ← M_{3i-1} 00 24: (Δ'₁, Δ'₂) ← F^{Δ₂⊕T, b}_{Δ₁⊕M_{3i}}(M_{3i-2}) 25: Δ ← Δ'₁[1...a] Δ'₂[1...a] ((Δ'₁ ⊕ 26: Δ'₂)[a+1...n]) 27: (Δ₁, Δ₂) ← (Δ[1...k], Δ[k+1...n+a]) 28: end for 29: T ← (P*[n+1...n+t-2]) l'pad 30: X, Y ← F^{Δ₂⊕T, b}_{Δ₁⊕P*[n+t-1... 31: n+t+k-2]}(P*[1...n]) 32: Z ← X[1...a] Y[1...a] 33: ((X ⊕ Y)[a+1...n]) 34: Tag ← Z[1...τ] 35: end function </pre>
<pre> 1: function Pad(x, y, M) // y ≤ x 2: l'pad ← 01; res ← M %x 3: if res ≠ y then 4: l'pad ← 11 5: if res < y then 6: M ← M 10^{y-1-res} 7: else 8: M ← M 10^{x+y-1-res} 9: end if 10: end if 11: return M, l'pad 12: end function 13: 14: function MAC(K, e, τ, M) 15: K₁, K₂ ← Derive(K) 16: M, l'pad ← Pad(n + t + k - e, 0, M) 17: P₁, ..., P_{ℓ-1} ← $\frac{n+t+k-e}{M}$ </pre>	<pre> 18: Δ₁, Δ₂, Δ₃ ← 0ⁿ 19: for i ← 1 to ℓ - 1 do 20: M_{3i-2} ← P_i[1...n] 21: M_{3i-1} ← P_i[n+1...n+t-e] 22: M_{3i} ← P_i[n+t-e+1...n+t+k-e] 23: T ← M_{3i-1} ⁽ⁱ⁾e-2 00 24: Δ₁ ← (F^{K₂⊕T, 0}_{K₁⊕M_{3i}}(M_{3i-2}) ⊕ Δ₁) ⊕_{t-2} 25: M_{3i-1} 26: Δ₂ ← 2 · (F^{K₂⊕T, 0}_{K₁⊕M_{3i}}(M_{3i-2}) ⊕ Δ₂) 27: Δ₃ ← M_{3i} ⊕ Δ₃ 28: end for 29: T ← Δ₁ l'pad 30: X, Y ← F^{K₂⊕T, b}_{K₁⊕Δ₃}(Δ₂) 31: Tag ← (X Y)[1...τ] 32: end function </pre>
<pre> 1: function Pad(a, b, M) // b ≤ a 2: l'pad ← 01; res ← M %a 3: if res ≠ b then 4: l'pad ← 11 5: if res < b then 6: M ← M 10^{b-1-res} 7: else 8: M ← M 10^{a+b-1-res} 9: end if 10: end if 11: return M, l'pad 12: end function 13: 14: function MAC(K, τ, N, M) 15: K₁, K₂ ← Derive(K) 16: M ← N M 17: M, l'pad ← Pad(n + t + k - 2, t + k - 2, M) 18: P₁, ..., P_{ℓ-1}, P* ← $\frac{n+t+k-2}{M}$ 19: (Δ₁, Δ₂) ← (0ⁿ, 0ⁿ) </pre>	<pre> 20: for i ← 1 to ℓ - 1 do 21: M_{3i-2} ← P_i[1...n] 22: M_{3i-1} ← P_i[n+1...n+t-2] 23: M_{3i} ← P_i[n+t-1...n+t+k-2] 24: T ← (M_{3i-1} ⊕_{t-2} Δ₂) 00 25: if i = 1 then 26: (Δ₂, Δ₁) ← F^{K₂⊕T, b}_{K₁⊕M_{3i}}(M_{3i-2}) 27: else 28: Δ ← Δ₂ 29: Δ₂ ← F^{K₂⊕T, 0}_{K₁⊕M_{3i}}(M_{3i-2} ⊕ Δ₁) 30: Δ₁ ← Δ 31: end if 32: end for 33: T ← (P*[1...t-2] ⊕_{t-2} Δ₂) l'pad 34: X, Y ← F^{K₂⊕T, b}_{K₁⊕P*[t-1...t+k-2]}(Δ₁) 35: Tag ← (X Y)[1...τ] 36: return Tag 37: end function </pre>

Figure 8: MAC algorithm/pseudocode of BabySonic_{n/a} (top), SuperSonic (center) and DarkSonic (bottom) mode. Here Derive is some secure key derivation function that is used to generate k -bit and $(t-2)$ -bit keys K_1 and K_2 , respectively.