Sonikku: Gotta Speed, Keed! A Family of Fast and Secure MACs

Amit Singh Bhati[1,2], Elena Andreeva[3], Simon Müller[3], and Damian Vizár[4]

[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. Message authentication codes (MACs) are fundamental symmetric key cryptographic functions used to generate a short, secret-key-dependent tag for a given message. This tag ensures both message authenticity and integrity, as computing a valid tag without the secret key is computationally infeasible, thereby revealing any unauthorized modification.

Existing MACs often rely on block ciphers (BCs) and tweakable block ciphers (TBCs). The design of these MACs involves various trade-offs regarding properties such as data processing rate, the number of secret keys, achievable security definitions and concrete margins, the necessity for pre- or post-processing, parallelization capabilities, internal state size, and performance optimization for diverse message lengths.

This work introduces Sonikku, a new family of MACs based on expanding primitives, comprising three distinct instances: BabySonic, DarkSonic, and SuperSonic. The Sonikku MACs offer a compelling combination of advantages: 1) superior speed compared to state-of-the-art TBC-based MACs; 2) security beyond the birthday bound related to the input block size; 3) a smaller internal state than comparable contemporary MACs; and 4) design flexibility considering diverse trade-offs, including pre/post-processing-free operation, parallel processing, a small resource footprint, and suitability for both short and long messages. These characteristics make them highly attractive for widespread applications, including resource-constrained environments like IoT and embedded devices.

Performance evaluations on a Cortex-M4 32-bit microcontroller demonstrate that BabySonic instantiated with ForkSkinny achieves a significant speed-up of at least 2.11x (and up to 4.36x) compared to the state-of-the-art ZMAC instantiated with SKINNY for 128-bit block sizes and messages up to 95 bytes. Similarly, DarkSonic and SuperSonic instantiated with ForkSkinny exhibit speed improvements of at least 1.93x for short messages (up to 95 bytes) and 1.48x for larger messages (up to 64KB), respectively, when benchmarked against ZMAC instantiated with SKINNY for both 64- and 128-bit block sizes.

Building upon the approach of ZMAC and PMAC2x, we further illustrate the potential of the Sonikku family by employing SuperSonic to

construct SonicAE, a highly efficient, beyond-birthday secure, stateless, and deterministic authenticated encryption scheme.

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 cryptographic tools ensuring authenticity and integrity in two-party secret key communication. The sender computes a tag over the message using a MAC algorithm and a shared secret key K. Upon receiving the message-tag pair, the receiver verifies the message's authenticity and integrity using the same MAC algorithm and key K.

In recent years, numerous MACs based on block ciphers (BCs) and tweakable block ciphers (TBCs) have been proposed. Notable examples include PMAC [15], the NIST standard CMAC [23], the ISO/IEC standard LightMAC [31], and their optimized successors such as PMAC1 [37], PMAC_Plus [43], 1k-PMAC_Plus [19], PMAC_TBC1k [32], and the state-of-the-art (SotA) MACs PMAC2x [29] and ZMAC [24].

Many of these MACs offer birthday-bound (BB) security guarantees relative to the block size n. Specifically, they achieve $n/2$-bit security. While (T)BCs with small block sizes ($n = 64$) have a smaller footprint and are better suited for constrained environments compared to those with larger blocks ($n = 128$), using BB-secure MACs with $n = 64$ is often deemed impractical due to feasible attacks [10]. This makes MACs offering security beyond the birthday bound (BBB) particularly desirable for lightweight platforms.

(T)BC-based MACs can be broadly categorized into sequential and parallel processing types. Sequential MACs typically have a smaller footprint and can avoid post-processing/finalization call overheads, making them suitable for memory-constrained applications processing primarily short messages. Parallel MACs enable full parallelization, offering significant speedups with hardware acceleration on multi-core platforms. They can also benefit from incremental processing [7], where modifying a few message bits/blocks does not necessitate recomputing the entire tag for long messages, reducing update costs. This property is valuable when the tag is frequently generated and updated.

Furthermore, (T)BC-based MACs can be nonce-based [16], serving as crucial components not only for authentication but also within authenticated encryption (AE) schemes.

Designing MACs suitable for low-cost, low-power IoT or embedded devices often requires balancing conflicting constraints, especially without dedicated cryptographic hardware. Factors such as block size (large for security, small for area), state size, number of primitive calls, and the need for pre/post-processing must be carefully considered to optimize performance for varying message lengths while maintaining adequate security.

For example, in industrial heavy machinery remote control, authentication data is typically a few bytes (e.g., button states, joystick position) transmitted via optimized protocols. The communication round trip demands stringent low latency (often < 100 ms [39,40,44]). Minimizing tag computation time is critical in such scenarios.

Another similar example is Automotive V2X (Vehicle-to-Everything) Communication that requires authentication for safety-critical messages such as collision warnings or traffic updates. These messages are often short, require low-latency processing, and must be highly secure against forgery.

Conversely, longer messages requiring authentication might include large binary files like multimedia content for embedded devices (KB for pictures, MB for videos). On low-end microcontrollers, content processing can consume significant resources, leaving limited time for MAC computation, particularly without hardware acceleration.

While using an expanding primitive for authentication might initially seem counterintuitive, this work demonstrates that MACs based on tweakable expanding primitives like forkciphers [4], such as ForkSkinny [35], can overcome traditional limitations in lightweight MAC design. A forward forkcipher evaluation is computationally less expensive than two parallel (T)BC calls with independent keys, yet yields two outputs. ForkSkinny, for instance, achieves favorable performance-security trade-offs by forking its internal SKINNY state partway through execution.

Expanding primitives have previously been shown to be useful in constructing cryptographic schemes with optimized performance and/or security for various applications, including authenticated encryption with associated data (AEAD) for short messages [4] and with leakage resilience [9,17]; robust online AEAD [2,3, 11]; transciphering-friendly AEAD schemes [13]; efficient and beyond BB-secure confidentiality-only encryption [1]; length-preserving encryption [14]; pseudo random number generators [5]; and key derivation functions [12].

Our Contribution. In this work, we introduce the Sonikku MAC family, based on expanding primitives. This family comprises two sequential MACs, BabySonic and DarkSonic, and one fully parallelizable MAC, SuperSonic. Sonikku MACs achieve strong variable-input-length (VIL) pseudorandom function (PRF) security, ranging from beyond birthday bound (BBB) up to full n-bit levels. DarkSonic and SuperSonic achieve PRF security when the forkcipher with a random and secret key is modeled as a pair of two tweakable random permutations, corresponding to the standard *pseudorandom tweakable forked permutation* (prtfp) definition [4]. Furthermore, all Sonikku members achieve PRF security under our extended prtfp definition for an XOR-related-tweakey setting, termed xrtk-prtfp. All Sonikku instances 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, while BabySonic achieves $3n/4$-bit security.

In addition, our MACs significantly improve upon State-of-the-Art (SotA) TBC-based MACs in terms of performance and flexibility. Compared to ZMAC, the Sonikku members offer the following advantages:

1. Require significantly lower number of primitive calls (at least 33% fewer calls).
2. More concretely, on a 32-bit Cortex-M4 processor with $n = 64$ (or 128), they save at least 40% (or 34%) in clock cycles for any message size. For short messages ($\leq$ 48B), BabySonic with $n = 128$ can save up to 77% in clock cycles.
3. Include pre- and/or post-processing free options, optimizing them for both short and long queries.
4. SuperSonic requires comparable state size whereas the other two Sonics require strictly smaller state sizes for any given security level.

For a detailed comparison of Sonikku MACs with relevant existing (T)BC-based MACs, we refer the reader to Table 1. Performance comparisons of Sonikku MACs instantiated with ForkSkinny against SotA ZMAC and PMAC2x instantiated with SKINNY are shown in Fig. 5 and Table 2. For simplicity, in the remainder of this work, Π[E] denotes the MAC function Π instantiated with the primitive E.

BabySonic[ForkSkinny] achieves a speed-up of at least **2.11x** (and up to **4.36x**) over ZMAC[SKINNY] for $n = t = k = 128$ (where k is the key size) for messages up to 95B. DarkSonic[ForkSkinny] and SuperSonic[ForkSkinny] achieve speed-ups of at least **1.93x** and **1.48x** over ZMAC[SKINNY] for $k = 128$ and $t = n \in \{64, 128\}$ for short messages up to 95B and large messages up to 64KB, respectively.

Finally, we propose an AEAD scheme called SonicAE based on the SIV-type composition of SuperSonic and the GCTR_2-3 [1] encryption scheme (Sec. 5). Our SonicAE mode provides the strong MRAE security guarantee [38]. Performance comparisons of SonicAE with other SotA AEAD schemes are provided in Fig. 7 and Table 3. SonicAE[ForkSkinny] achieves a speed-up of at least **1.41x** and **1.32x** over existing MRAE schemes ZAE[SKINNY] and Deoxys-II[SKINNY] under $k = 128$, $t = n \in \{64, 128\}$ for short messages up to 95B and large messages up to 64KB, respectively.

Related Work. Datta et al. in [18] proposed LightFORK, a forkcipher variant of LightMAC that achieves beyond birthday security. However, in contrast to our parallel MAC SuperSonic, which can process at least $n + k$ bits in the standard single-key setting (and up to $n + t + k - e$ bits in the XRTK setting) per "*one-legged*" forkcipher call, LightFORK processes only n bits per "*two-legged*" forkcipher call. This makes LightFORK significantly costlier than SuperSonic, specifically at least **3x** costlier for $k \geq n$.

Paper Organization. Sec. 2 introduces necessary notations and security notions. Sec. 3 formally describes the Sonikku family and its security results, with proofs deferred to App. A. Sec. 4 discusses our Sonikku results and software per-

MAC	Primitive	PRF Security (IT)	$\|\mathcal{K}\|$	Minimal State	Cost [5](to process an m-bit message)	$\|Tag\|$	L.I.Sec.	Par.
CMAC [23]	BC as prp [28]	$n/2$	k	$3n+k$	$(\lceil m/n \rceil + 1)$BC + 2BM	$\leq n$	✗	✗
PMAC [15]	BC as prp	$n/2$	k	$3n+k$	$(\lceil m/n \rceil + 1)$BC + $\lceil m/n \rceil$BM	$\leq n$	✗	✓
PMAC1 [37]	TBC as tprp [28]	$n/2$	k	$2n+t+k$	$\lceil m/n \rceil$TBC	$\leq n$	✗	✓
lightMAC [31]	BC as prp	$n/2$	$2k$	$3n+2k$	$\lceil m/(n-s) \rceil$BC	$\leq n$	✓	✓
SUM-ECBC [42]	BC as prp	$2n/3$	$4k$	$2n+4k$	$2(\lceil m/n \rceil + 1)$BC	$\leq n$	✗	✗
PMAC_Plus [43]	BC as prp	$2n/3$	$3k$	$5n+3k$	$(\lceil m/n \rceil + 4)$BC + $(3\lceil m/n \rceil$-1)BM	$\leq n$	✗	✓
1k-PMAC_Plus [19]	BC as prp	$2n/3$	k	$5n+k$	$(\lceil m/n \rceil + 4)$BC + $3\lceil m/n \rceil$BM	$\leq n$	✗	✓
ZMAC [24]	TBC as tprp	$\min\{n,(n+t)/2\}$	k	$4n+2t+k$	$\left(\left\lceil \frac{m}{n+t-4} \right\rceil + 6\right)$TBC + $\left(2\left\lceil \frac{m}{n+t-4} \right\rceil - 1\right)$BM	$\leq 2n$	✗	✓
PMAC_TBC1k [32]	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	$\leq n$	✓	✓
PMACx/PMAC2x [29]	TBC as tprp	n	k	$3n+t+k$	$(\lceil m/n \rceil + 2)$TBC + $\lceil m/n \rceil$BM	$\leq 2n$	✓	✓
BabySonic [Our Work]	FC as xrtk-prtfp	$3n/4$	k	$2n+t+k$	$\left\lceil \frac{m}{n+t+k-2} \right\rceil$FC	$\leq 1.5n$	✗	✗
DarkSonic [Our Work]	FC as xrtk-prtfp	$\min\{n,t\} - \log_2 \mu$	k	$2n+t+2k$	$\left(\left\lceil \frac{m-2(t+k-2)}{n+t+k-2} \right\rceil\right)$TBC + 2FC	$\leq 2n$	✗	✗
SuperSonic [Our Work]	FC as xrtk-prtfp	$\min\{n,(n+t)/2\}$	k	$2n+2t+3k$	$\left(\left\lceil \frac{m}{n+t+k-e} \right\rceil\right)$(TBC + BM) + 1FC	$\leq 2n$	✓	✓

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 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. $|\mathcal{K}|$ and $|Tag|$ represent the key length and the maximum possible tag length of the MAC, respectively. For DarkSonic and SuperSonic, the table rows remain correct even if the parts in blue are removed, i.e. the security still holds, potentially with some reduced performance. The cells in red highlight the key points of comparison against Sonikku modes.

formance. Sec. 5 presents SonicAE as an application of SuperSonic for misuse-resistant authenticated encryption. Sec. 6 concludes the paper.

2 Preliminaries

2.1 Notation

Strings and Operations. All strings are considered 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 $\{0,1\}^n$. For a string X of ℓ bits, $X[i]$ denotes the i-th bit of X for $i = 1, \ldots, \ell$ (counting from left to right), and $X[i \ldots j] = X[i] \| X[i+1] \| \ldots \| X[j]$ for $1 \leq i < j \leq \ell$. For two strings $X, Y \in \{0,1\}^*$ with $|X| \leq |Y|$ without loss of generality, $X \oplus Y$ denotes the bitwise XOR of $X \| 0^{|Y|-|X|}$ and Y. For the same strings, $X \oplus_a Y = (X \oplus Y)[1 \ldots a]$.

For an arbitrary integer n, which we fix as the block size for this work, we define the partitioning of a string X into n-bit blocks as $X = X_1 \| \ldots \| X_x \| X_*$,

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

where $|X_i| = n$ for $i = 1, \ldots, x$ and $0 < |X_*| \leq n$. Here, $x = \lceil |X|/n \rceil - 1$ represents the number of full n-bit blocks before the last block. For two distinct strings $X, Y \in \{0,1\}^*$, we define $\mathsf{llcp}_n(X, Y)$ as the number of common full n-bit blocks at the beginning of X and Y. Formally, $\mathsf{llcp}_n(X, Y) = i$ if $X_j = Y_j$ for all $1 \leq j \leq i$ and either $X_{i+1} \neq Y_{i+1}$ or X or Y ends after block i. If $X_1 \neq Y_1$, then $\mathsf{llcp}_n(X, Y) = 0$.

Adversary Interaction. An adversary, denoted by $\mathcal{A}$, is a probabilistic algorithm. Its function involves interacting with oracles, which are indicated by superscripts. The adversary's ultimate output is either 0 or 1. The notation $\mathcal{A}^{\mathcal{O}} \Rightarrow 1$ specifically describes the scenario where the adversary $\mathcal{A}$ makes queries to oracle $\mathcal{O}$ during its process and subsequently produces an output of 1.

Miscellaneous. $X \leftarrow\!\!\$\ \mathcal{X}$ denotes the sampling of an element X from a finite set $\mathcal{X}$ according to the uniform distribution. $(p)_q$ denotes the falling factorial $p(p-1)\ldots(p-q+1)$, with $(p)_0 = 1$. A predicate $\mathsf{P}(x)$ is defined as $\mathsf{P}(x) = 1$ if it is true and $\mathsf{P}(x) = 0$ if it is false. Integer tuples (i', j') are compared lexicographically, i.e., $(i', j') < (i, j)$ if and only if $i' < i$ or ($i' = i$ and $j' < j$).

2.2 MAC Syntax and Security Definition

Let $\mathcal{X}$ and $\mathcal{Y}$ be non-empty finite sets. Let $\mathrm{Func}(\mathcal{X}, \mathcal{Y})$ 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 $R : \mathcal{X} \to \mathcal{Y}$, is a random function sampled uniformly from $\mathrm{Func}(\mathcal{X}, \mathcal{Y})$.

A MAC is defined by a tuple: $(\mathcal{K}, \mathsf{MAC}, \mathsf{Verify})$, where $\mathcal{K}$ is the non-empty key space, $\mathsf{MAC} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is the tag generation function, and $\mathsf{Verify} : \mathcal{K} \times \mathcal{X} \times \mathcal{Y} \to \{1, \perp\}$ is the tag verification function. For any key $K \in \mathcal{K}$ and message $X \in \mathcal{X}$, $\mathsf{Verify}_K(X, Y)$ returns 1 if and only if $\mathsf{MAC}_K(X) = Y$, and $\perp$ otherwise. The verification function is typically defined as recomputing the tag and checking for equality; thus, for brevity, we refer to a MAC scheme by the tuple $(\mathcal{K}, \mathsf{MAC})$ in the remainder of this paper. A secure pseudorandom function (PRF) is known to imply a secure MAC; however, the converse is not necessarily true. We recall the standard definition of PRF security for a MAC.

Definition 1 (PRF Advantage). *For $\mathsf{MAC} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$, let $\mathcal{A}$ be an adversary whose goal is to distinguish $\mathsf{MAC}(K, \cdot)$ from a URF $R(\cdot) : \mathcal{X} \to \mathcal{Y}$ via oracle access. The PRF advantage of $\mathcal{A}$ against MAC is defined as:*

$$\mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{MAC}}(\mathcal{A}) = \left| \Pr[K \leftarrow^{\$} \mathcal{K} : \mathcal{A}^{\mathsf{MAC}(K,\cdot)} \Rightarrow 1] - \Pr[R \leftarrow^{\$} \mathrm{Func}(\mathcal{X}, \mathcal{Y}) : \mathcal{A}^{R(\cdot)} \Rightarrow 1] \right|.$$

2.3 Tweakable Expanding Primitives

A tweakable expanding primitive $\mathsf{F} : \{0,1\}^k \times \{0,1\}^t \times \{0,1\}^n \to \{0,1\}^{\alpha n}$ for $\alpha \geq 2$ maps a k-bit key K, a t-bit tweak T, and an n-bit input M to α output blocks $Y_1, \ldots, Y_\alpha$, each of n bits. When all α outputs are permutations of the input for any fixed key and tweak, F_α is called a multiforkcipher (MFC) [1]. In this work, we focus on $\alpha = 2$, referring to such primitive as forkcipher (FC) [4]. We denote a forkcipher as F. The parameters k, n, and t are its key size, block

size, and tweak size, respectively. We use the notations $\mathsf{F}(K, T, M)$, $\mathsf{F}_K(T, M)$, or $\mathsf{F}_K^T(M)$ interchangeably. The output $Y_1 \| Y_2 \in \{0,1\}^{2n}$.

We use an additional parameter $s \in \{0, 1, b\}$ to specify which output(s) are returned from an F call: $\mathsf{F}_K^{T,s}(M) := \mathsf{F}_K^T(M)[1 \dots n]$ when $s = 0$ (left output), $\mathsf{F}_K^T(M)[n+1 \dots 2n]$ when $s = 1$ (right output), and $\mathsf{F}_K^T(M)$ when $s = b$ (both outputs). For the rest of the paper, we use $\mathcal{K} := \{0,1\}^k$ and $\mathcal{T} := \{0,1\}^t$.

2.4 XOR-Related-Tweakey (XRTK) Security

Security of cryptographic constructions is typically analyzed in the Standard Model, where primitives are treated as fixed algorithms with secret random keys. More idealized models, like the Ideal Cipher Model (ICM), simplify analysis by treating primitives as truly random permutations for *any* adversarially chosen key.

We introduce the XOR-Related-Tweakey (XRTK) model, positioned between the Standard Model and the ICM. In this model, the adversary does not know the secret random base key $K_{base} \leftarrow\$ \{0,1\}^k$ and tweak $T_{base} \leftarrow\$ \{0,1\}^t$. However, the adversary can query the primitive F with inputs $(T_{base} \oplus \Delta T, K_{base} \oplus \Delta K, M)$, where $\Delta T \in \{0,1\}^t$ and $\Delta K \in \{0,1\}^k$ are chosen by the adversary. This model captures scenarios where keys or tweaks might be updated via XOR operations with secret state, which is specially relevant for feedback-based MAC designs. It grants the adversary more power than the Standard Model (where only queries under a single secret key are allowed) but is weaker than the ICM, which assumes ideal behavior for arbitrary, potentially unrelated, adversarially chosen keys.

Formally, the XRTK *pseudorandom tweakable forked permutation* ($\mathsf{xrtk\text{-}prtfp}$) security of a forkcipher F is defined as the indistinguishability between the real ($\mathbf{xrtk\text{-}prtfp}_\mathsf{F}^{\text{real}}$) and ideal ($\mathbf{xrtk\text{-}prtfp}_\mathsf{F}^{\text{ideal}}$) worlds by an adversary making chosen-plaintext queries.

In the real world ($\mathbf{xrtk\text{-}prtfp}_\mathsf{F}^{\text{real}}$), depicted in Fig. 1 (left), the oracle uses secret random bases $K_1 \leftarrow\$ \{0,1\}^t$ and $K_2 \leftarrow\$ \{0,1\}^k$. For an adversary query $(\mathsf{T}_1, \mathsf{T}_2, M, s)$, the oracle computes $T = K_1 \oplus \mathsf{T}_1$ and $K' = K_2 \oplus \mathsf{T}_2$, evaluates $\mathsf{F}_{K'}(T, M)$, and returns the specified output(s) according to $s \in \{0, 1, b\}$. Note that the adversary's input $(\mathsf{T}_1, \mathsf{T}_2)$ directly corresponds to the XOR differences $(\Delta T, \Delta K)$ applied to the secret bases.

In the ideal world ($\mathbf{xrtk\text{-}prtfp}_\mathsf{F}^{\text{ideal}}$), depicted in Fig. 1 (right), the oracle is a collection of independent random permutations $\pi_{V,0}, \pi_{V,1} \leftarrow\$ \text{Perm}(n)$ for each $V \in \{0,1\}^{t+k}$. For an adversary query $(\mathsf{T}_1, \mathsf{T}_2, M, s)$, the oracle sets $V = \mathsf{T}_1 \| \mathsf{T}_2$, evaluates $(\pi_{V,0}(M), \pi_{V,1}(M))$, and returns the specified output(s) according to s. The $\mathsf{xrtk\text{-}prtfp}$ advantage of an adversary $\mathcal{A}$ is defined as

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

This notion provides a strong security guarantee for primitives used in constructions where the effective key/tweak can be modeled as a secret base XORed with public or state-dependent values. $\mathsf{xrtk\text{-}prtfp}$ reduces to its single-key counterpart; the standard prtfp [4] when $\Delta K = 0$.

Game $\mathbf{xrtk\text{-}prtfp\text{-}real}_{\mathsf{F}}$	Game $\mathbf{xrtk\text{-}prtfp\text{-}ideal}_{\mathsf{F}}$
$f_1 \leftarrow^{\$} \mathrm{Func}(k,t), f_2 \leftarrow^{\$} \mathrm{Func}(k,k)$	**for** $\mathsf{T}_1 \| \mathsf{T}_2 \in \{0,1\}^{t+k}$ **do**
$K \leftarrow^{\$} \mathcal{K}, K_1 \leftarrow f_1(K), K_2 \leftarrow f_2(K)$	$\pi_{\mathsf{T}_1\|\mathsf{T}_2,0}, \pi_{\mathsf{T}_1\|\mathsf{T}_2,1} \leftarrow^{\$} \mathrm{Perm}(n)$
$b \leftarrow \mathcal{A}^{\mathcal{E}}$	$b \leftarrow \mathcal{A}^{\mathcal{E}}$
return b	**return** b
Oracle $\mathcal{E}(\mathsf{T}_1, \mathsf{T}_2, X)$	**Oracle** $\mathcal{E}(T_1, T_2, X)$
return $\mathsf{F}_{K_2 \oplus \mathsf{T}_2}(K_1 \oplus \mathsf{T}_1, X)$	**return** $\pi_{T_1\|T_2,0}(X), \pi_{T_1\|T_2,1}(X)$

Fig. 1: xrtk-prtfp security games for a forkcipher F. The adversary queries $(\mathsf{T}_1, \mathsf{T}_2, M, s)$, representing chosen XOR differences $\Delta T = \mathsf{T}_1$ and $\Delta K = \mathsf{T}_2$. The real oracle uses secret random bases $K_1 \leftarrow\$ \{0,1\}^t, K_2 \leftarrow\$ \{0,1\}^k$ as base tweak $T_{base} = K_1$ and base key $K_{base} = K_2$. The ideal oracle uses random permutations indexed by $\mathsf{T}_1\|\mathsf{T}_2$.

Justification for the XRTK Model. The xrtk-prtfp notion is a powerful yet realistic assumption for modern primitives like ForkSkinny. While weaker than the ideal cipher assumption (which implies ideal behavior for *any* key/tweak choice), it significantly strengthens the Standard Model assumption by allowing adversarial control over XOR differences relative to secret bases. Unlike the ICM, xrtk-prtfp security can potentially be argued based on the specific design of the primitive and its key/tweak schedule. The feasibility of this assumption for ForkSkinny is supported by the extensive cryptanalysis of SKINNY and the general tweakey framework under related-key and related-tweakey models [6, 20, 30, 36, 45]. These analyses provide evidence that ForkSkinny's design resists attacks that exploit XOR relations in keys and tweaks, making the xrtk-prtfp assumption a well-justified basis for the security of our MAC constructions in this work.

Further, the generic attack results of [8][Lemma 5.3, 5.4, Theorem 6.3 and 8.8] against an ideal primitive suggest security for a forkcipher in XOR-related setting up to birthday bound in the effective key length i.e., up to $\mathcal{O}(2^{\frac{t+k}{2}})$ queries. For ForkSkinny, where $t + k \geq 2n$, this translates to n bits of security against generic attacks, aligning with our beyond-birthday security claims for FC-based MACs.

3 Sonikku Family of Fast and Secure MACs

The Sonikku family comprises three fast and secure MACs based on tweakable expanding primitives: BabySonic, SuperSonic, and DarkSonic. All Sonikku instances provably achieve Beyond Birthday Bound (BBB) PRF-security and demonstrate significant speed improvements compared to state-of-the-art (SotA) MACs in

various application settings (see Sec. 4). To achieve this speed-up, Sonikku MACs effectively utilize the input space, tweak space, and importantly, the key space of the underlying primitive to securely combine message blocks with available key material.

Design Choices. The Sonikku MACs offer distinct trade-offs in their design. SuperSonic is designed for hardware parallelization and fast performance on long messages. It employs parallel primitive calls for message processing and maintains an extra internal state to accumulate chained values for a final primitive evaluation (the finalization call), as detailed in Fig. 3.

In contrast, DarkSonic and BabySonic are sequential MACs. They sacrifice hardware parallelization support to reduce the minimal state size (the amount of required RAM), making them more suitable for platforms without parallel processing capabilities or with stringent resource constraints.

BabySonic minimizes state by using the key only during an initial setup phase, eliminating the need to store the full key in the main processing loop's state. This results in an optimal minimal state size, equivalent to the state size of the underlying expanding primitive. BabySonic utilizes both outputs of the expanding primitive (F) to generate an internal pseudorandom state, which helps eliminate dedicated pre- or post-processing calls (such as the initial call in DarkSonic or the final call in SuperSonic). This design optimizes BabySonic for short message queries, particularly those up to 95 bytes.

DarkSonic, on the other hand, is a nonce-based construction. It achieves improved performance with increasing message length (approaching SuperSonic's speed for various message sizes when there is no parallel processing) by utilizing only one output branch in most F calls. However, this comes at the cost of requiring a nonce input for security (see Table 1).

Our Recommendation. Based on their performance characteristics across message lengths (illustrated in Fig. 5(b)), we recommend BabySonic for short messages (up to approximately 136 bytes) and SuperSonic for longer messages (above approximately 96 bytes). DarkSonic serves as a valuable option with intermediate trade-offs, particularly where a nonce is available and efficient processing of longer sequential messages is needed.

The following subsections formally define each Sonikku instance and state their PRF security results. Detailed proofs are provided in App. A.

3.1 BabySonic and its PRF Security

$\mathsf{BabySonic}_{n/a}$ is a sequential MAC utilizing a tweakable expanding primitive F with tweak space $\mathcal{T} = \{0,1\}^t$ for $t > 0$. $\mathsf{BabySonic}_{n/a}[\mathsf{F}] := (\mathcal{K}, \mathsf{MAC})$ has key space $\mathcal{K} = \{0,1\}^k$ and message space $\mathcal{M} = \{0,1\}^*$. It offers an "*optimal*" internal state size, meaning the memory required for the internal state during processing is equal to the minimal state size of the underlying expanding primitive itself, incurring no state overhead from the MAC construction. The MAC algorithm for $\mathsf{BabySonic}_{n/a}$ is depicted in Fig. 2 and its pseudocode is given in Fig. 8. Its PRF security is formally stated in Theorem 1, with its proof deferred to App. A.1.

Theorem 1. *Let* F *be a forkcipher 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 to* $\mathsf{BabySonic}_{n/a}$*, with maximum query length* ℓ *(in blocks), such that the total number of* F *calls induced is at most* σ *and* $10 \leq a \leq \min\{n - 10, 11n/12\}$*, we have*

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

for some adversary $\mathcal{A}'$ *making at most* q F *queries (under fixed but secret and random bases as per the* xrtk-prtfp *model), running in time bounded by the running time of* $\mathcal{A}$ *plus* $\gamma \cdot \sigma$*, where* γ *is the runtime of an* F *call.*

From this bound, $\mathsf{BabySonic}_{n/a}$ achieves a maximum information-theoretic PRF security of approximately $3n/4$ bits (with respect to primitive queries) when $a = n/2$. Thus, $\mathsf{BabySonic}_2$ emerges as the optimal variant of $\mathsf{BabySonic}_{n/a}$. For simplicity, this variant is referred to as BabySonic throughout the paper.

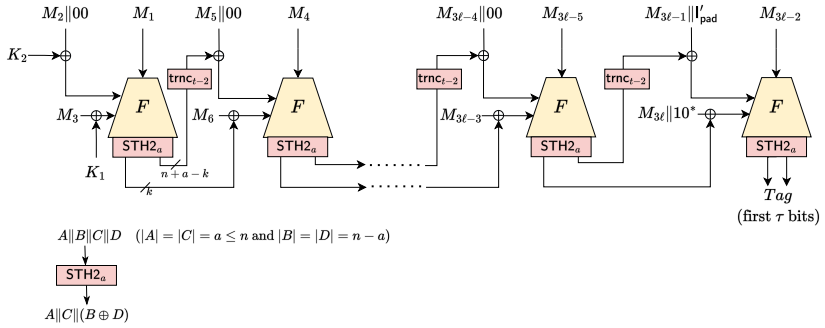


Fig. 2: The $\mathsf{BabySonic}_{n/a}$ MAC mode. Processing of a message M is illustrated. Padding of 10^* is applied to make the message size a multiple of $(n + t + k - 2)$ for the smallest possible positive integer x' blocks (see Fig. 8). The padding indicator $\mathsf{l}'_{\mathsf{pad}}$ is 01 if the original message size was a multiple of the block size $(n + t + k - 2)$, and 11 otherwise. The STH2 function (Summation-Truncation Hybrid-2 [22], shown below) processes the $2n$-bit F output to produce the $(n+a)$-bit internal chaining value (or tag), where $n+a \geq k$. The k-bit key K is used to derive sub-keys $K_1 \in \{0,1\}^k$ and $K_2 \in \{0,1\}^{t-2}$ using a key derivation function.

3.2 SuperSonic and its PRF Security

SuperSonic is a parallel MAC based on a tweakable expanding primitive F with tweak space $\mathcal{T} = \{0,1\}^t$. $\mathsf{SuperSonic}[\mathsf{F}] := (\mathcal{K}, \mathsf{MAC})$ has key space $\mathcal{K} = \{0,1\}^k$ and message space $\mathcal{M} = \{0,1\}^*$. The SuperSonic algorithm is illustrated in Fig. 3 and its pseudocode is in Fig. 8. The formal claim regarding the PRF security of SuperSonic is presented in Theorem 2, with its proof in App. A.2.

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

$$\mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{SuperSonic}[\mathsf{F}]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{xrtk\text{-}prtfp}}_{\mathsf{F}}(\mathcal{A}') + \frac{4q^2}{2^{n+\min\{n,t-2\}}}$$

for some adversary $\mathcal{A}'$ *making at most* σ F *queries (under fixed but secret and random bases as per the* xrtk-prtfp *model), running in time bounded by the running time of* $\mathcal{A}$ *plus* $\gamma \cdot \sigma$*, where* γ *is the runtime of an* F *call.*

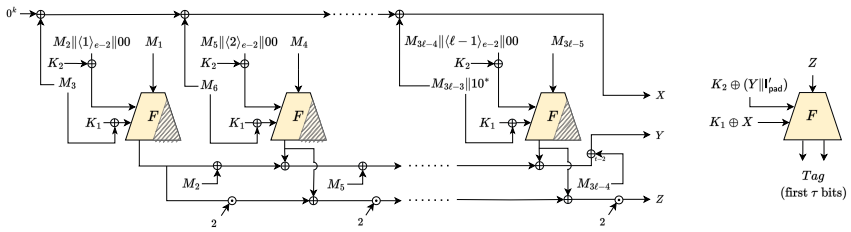


Fig. 3: The SuperSonic MAC mode. Processing of a message M is illustrated. Padding of 10^* is applied to make the message size the smallest possible multiple of $(n+t+k-e)$ blocks (see Fig. 8). The padding indicator $\mathsf{I}'_{\mathsf{pad}}$ is 01 if the original message size was a multiple of the block size $(n+t+k-e)$, and 11 otherwise. The k-bit key K is used to derive sub-keys $K_1 \in \{0,1\}^k$ and $K_2 \in \{0,1\}^{t-2}$ using a key derivation function. Note that e is a constant depending on the maximum message length (defined in Table 1).

3.3 DarkSonic and its PRF Security

DarkSonic is a nonce-based sequential MAC employing a tweakable expanding primitive F with tweak space $\mathcal{T} = \{0,1\}^t$. DarkSonic$[\mathsf{F}] := (\mathcal{K}, \mathcal{N}, \mathsf{MAC})$ has key space $\mathcal{K} = \{0,1\}^k$, nonce space $\mathcal{N} = \{0,1\}^n$, and message space $\mathcal{M} = \{0,1\}^*$. The DarkSonic algorithm is depicted in Fig. 4 and its pseudocode is in Fig. 8. Its PRF security (under nonce misuse setting) is formally stated in Theorem 3 with its proof deferred to App. A.3.

Theorem 3. *Let* F *be a forkcipher 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* F *calls induced is at most* σ *and the maximum number of times a nonce is repeated is* μ*, we have*

$$\mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{DarkSonic}[\mathsf{F}]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{xrtk\text{-}prtfp}}_{\mathsf{F}}(\mathcal{A}') + \frac{6(\sigma-q)^2}{2^{n+\min\{n,t-2\}}} + \frac{2q(\mu-1)}{2^{\min\{n,t-2\}}}$$

for some adversary $\mathcal{A}'$ *making at most* σ F *queries (under fixed but secret and random bases as per the* xrtk-prtfp *model), running in time bounded by the running time of* $\mathcal{A}$ *plus* $\gamma \cdot \sigma$*, where* γ *is the runtime of an* F *call.*

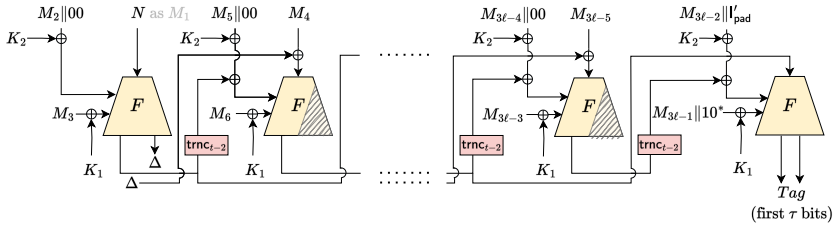


Fig. 4: The DarkSonic MAC mode. Processing of a full message M (including the n-bit nonce N) is illustrated. Padding of 10^* is applied to make the message size the smallest possible multiple of $(n+t+k-2)-n$ for some positive integer x' blocks, resulting in total length $(x'(n+t+k-2)-n)+n$ after appending the nonce (see Fig. 8). The padding indicator $\mathsf{I}'_{\mathsf{pad}}$ is 01 if the original message size was a multiple of the message block size $(n+t+k-2)-n$, and 11 otherwise. $\mathsf{trnc}_{t-2}(X)$ returns the first $t-2$ bits of $X\|0^{t-2}$. The k-bit key K is used to derive sub-keys $K_1 \in \{0,1\}^k$ and $K_2 \in \{0,1\}^{t-2}$ using a key derivation function.

Novelty. The Sonikku MACs introduce several novel aspects:

- BabySonic: This design is unique in utilizing key feedback to avoid storing the full key during the main processing loop, achieving an optimal minimal state size equal to that of the underlying primitive. It processes message blocks at a rate of 1 primitive call per block and achieves beyond birthday security. Unlike Nested-MAC (NMAC) [21]; a popular key feedback mode, which achieves a rate of $(n+t)/(n+t+k)$ but requires a FIL-PRF and offers security below the birthday bound in the key size k, BabySonic provides BBB security in the block size n. The key feedback structure necessitates a dedicated security proof technique involving iterative replacement of primitive calls with ideal components.
- DarkSonic: This is the first efficient nonce-based MAC (with nonce-misuse security) designed specifically around forkciphers or tweakable block ciphers. It uniquely achieves full n-bit security with only an n-bit chaining state (compared to the standard $2n$ bits in similar schemes) by securely re-using randomness from the previous state. The security analysis using the H-coefficient technique captures the complexities introduced by this novel state management.
- SuperSonic: While following the general structure of parallel BBB-secure MACs like PMAC_Plus, PMACx/2x, and ZMAC, SuperSonic introduces several innovations:
  1. It efficiently incorporates message blocks into the key argument of the internal primitive calls by maintaining a checksum of these blocks processed in a refined finalization step. Existing designs and their proofs do not directly support passing message blocks into the key argument in this manner.

2. It features a novel, efficient finalization call compared to SotA ZMAC. Specifically, it replaces four final TBC calls with a single FC call, which is particularly beneficial for short message processing.
3. It employs strategies such as "*counter-in-tweak*" and tweak domain separation to yield a simpler, less error-prone design with smaller long-term secret material, potentially reducing the cost of threshold implementations. The resulting security proof for SuperSonic is notably compact and simple, providing a length-independent bound.

TBC Variants. DarkSonic and SuperSonic can be instantiated with TBCs instead of FCs (replacing each branch of an FC with a tweak-domain separated TBC). These variants achieve security based on the related-key properties of the underlying TBC (by incorporating message blocks into the TBC's key input) and the security results transfer from the xrtk-prtfp model for FCs to related-key TBC models.

Switching to Single-Key Setting. DarkSonic's and SuperSonic's security results can also be 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.

Even when instantiated with TBCs and analyzed in a single-key setting (which typically reduces performance compared to utilizing related keys), DarkSonic and SuperSonic still demonstrate significant speed advantages over ZMAC. As shown in Table 2, for $n = 128$, these TBC variants in a single-key setting still achieve speed-ups of up to 1.6x and 1.77x for messages $\leq$ 95B, respectively. The speed-up converges towards 1x for very long messages ($\geq$ 64KB).

4 Performance and Discussion

We evaluated the performance of Sonikku MACs on a 32-bit Cortex-M4 processor (with no parallelization). Our fully parallelizable MAC, SuperSonic, offers length-independent security up to full n bits. When instantiated with ForkSkinny (and comparing against ZMAC instantiated with SKINNY) with $k = 128$ bits and $t = n \in \{64, 128\}$, SuperSonic significantly reduces the computation cost for tag generation. Specifically, it provides speed-ups of 2.76x (for $n = 64$) and 2.66x (for $n = 128$) for messages up to 16B (relevant for IoT/embedded devices), 1.93x (for $n = 64$) and 2.15x (for $n = 128$) for messages up to 95B (relevant for Noise framework [34] based protocols), and 1.58x (for $n = 64$) and 1.48x (for $n = 128$) for messages up to 64KB. Beyond performance, SuperSonic requires a minimal state up to $2n$ bits smaller than ZMAC in single-key setting (see Table 1).

Compared to PMAC_TBC1k and PMACx/PMAC2x, SuperSonic processes approximately $n + t + k$ bits of input per F call, whereas the mentioned schemes are limited to processing n bits per TBC call, regardless of tweak/key size. Table 1 provides a detailed comparison of Sonikku modes with existing state-of-the-art MACs.

The sequential MACs, DarkSonic and BabySonic, are well-suited for applications lacking parallel processing support or facing severe resource constraints like small, fixed state size. BabySonic excels by using the key only during initialization, thus eliminating the need for dedicated key storage state and achieving optimal minimal state equal to the primitive's state size. DarkSonic, a nonce-based MAC, utilizes only one output leg of the F calls, achieving significantly better performance than BabySonic for longer messages while offering comparable security (albeit nonce-based).

ZMAC was originally recognized as achieving optimal efficiency/rate among TBC-based MACs by processing $n + t$ bits of data per TBC call [24], assuming a TBC could handle n-bit input and t-bit tweak per call. Our work pushes this boundary by exploiting the k-bit key argument space of a tweakable primitive (particularly FCs) in each of the Sonikku MACs to incorporate an additional k-bit of public input per call, thereby resetting the margin for achievable optimality towards $n + t + k$ bits per call.

It is worth noting that introducing public inputs into the key argument of a traditional (tweakable) cipher is generally discouraged due to potential security vulnerabilities (e.g., related-key attacks) and efficiency concerns (e.g., key schedule recomputation). However, this approach is viable for Sonikku because: 1) Our security analysis relies on the well-defined xrtk-prtfp model (Sec. 2), which specifically accounts for XOR-related keys and tweaks; and 2) Modern lightweight primitives like SKINNY and ForkSkinny feature lightweight key schedules where recomputation cost is minimal, or benefit from hardware parallelism where key schedule updates are for free.

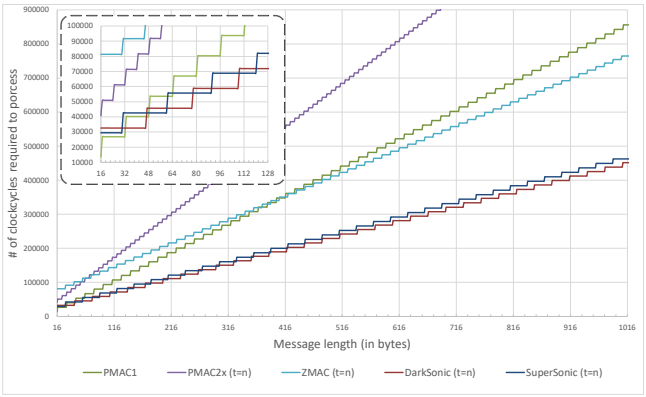
The feasibility of using primitives in an XOR-related-tweakey manner is supported by concrete security analyses and results on the tweakey framework [25] and the related-tweakey security of SKINNY [6] and ForkSkinny [4] as discussed in Sec. 2.

Fig. 5 provides an efficiency comparison of Sonikku MACs (instantiated with ForkSkinny) against SotA TBC-based MACs (instantiated with SKINNY) for $n = k = 128$. The plots show that Sonikku MACs require significantly fewer clock cycles than ZMAC and PMAC2x. Among the Sonikku family, DarkSonic and SuperSonic perform similarly for longer messages and outperform BabySonic due to their use of one-legged F calls in the main loop. For short messages, BabySonic is faster as it avoids pre/post-processing calls. Table 2 provides a detailed speed-up comparison against ZMAC for both $n = 64$ and $n = 128$ settings, confirming these observations.

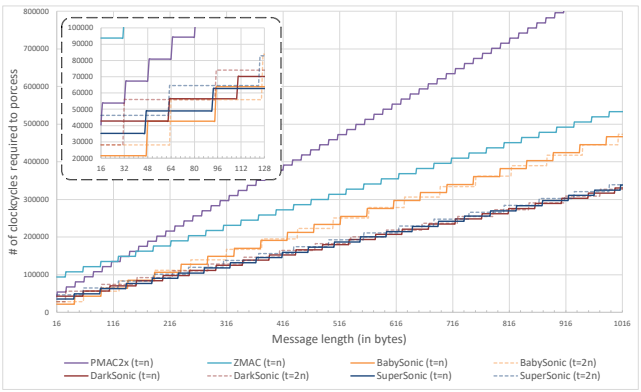
Our data also indicates that increasing the tweak size t can improve a Sonic MAC's performance, but this benefit is most pronounced for messages exceeding a certain length threshold (approximately 2KB for Cortex-M4).

5 SonicAE: SuperSonic based Deterministic AE

Following the design paradigms of SIVx [29] (based on PMAC2x) and ZAE [24] (based on ZMAC), we construct SonicAE, a stateless or deterministic authen-



(a) MACs targeting 64 bits of security



(b) MACs targeting 96 to 128 bits of security

Fig. 5: Efficiency comparison of Sonikku with other (tweakable) primitive based SotA MACs, using $k = 128$ and $n = 64$ or 128 bits depending on the targeted security. All TBCs and FCs are instantiated with SKINNY and ForkSkinny, respectively. The plots show the number of clock cycles required to process messages of corresponding lengths.

ticated encryption (DAE) scheme. A DAE scheme provides security that does not rely on inputs such as random IVs or unique nonces for its core security guarantees [38]. The security notion for DAE is known as dae. Importantly, a scheme proven dae-secure can accept a nonce as part of the associated data (AD) and inherently provides security against nonce misuse. Such schemes and their achieved security are referred to as misuse-resistant AEs (MRAEs) and the mrae notion, respectively. SonicAE is dae-secure and thus provides mrae security.

SonicAE is an SIV [38]-like composition utilizing the SuperSonic MAC (from Sec. 3.2) and a modified GCTR_2-3 [1] encryption mode, denoted GCTR'_2-3. The composition is illustrated in Fig. 6. GCTR'_2-3 is based on GCTR_2-3 but incorporates two key modifications:

1. Tweak domain separation is achieved by fixing the last two tweak bits to 10.

MAC →	BabySonic	DarkSonic		SuperSonic	
Block size n (bits) →	128	64	128	64	128
IT PRF Security (bits) →	96	64	128	64	128
Max. message length ↓	Speed-up (factor) 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

Table 2: Efficiency comparison of Sonikku (instantiated with ForkSkinny) against ZMAC (instantiated with SKINNY) with $k = 128$ bits and $t = n \in \{64, 128\}$ bits. Entries show the ratio of clock cycles for ZMAC relative to the corresponding Sonikku MAC for a given maximum message length. Green cells indicate where Sonikku is faster than ZMAC, while red indicates it is slower.

2. The nonce input used in GCTR_2-3 is replaced by the second n-bit half of the $2n$-bit tag generated by SuperSonic.

SonicAE is a dae-secure scheme, achieving beyond birthday security of $\min\{n, (n+t)/2\}$ bits (as stated in Theorem 4). It processes, on average, $2n(n+t+k)/(3n+t+k)$ bits per primitive call of the underlying expanding primitive F. For minimal settings of $t = 0$ and $k = n$, this processing rate is n bits per F call, comparable to existing FC-based AE schemes like PAEF, SAEF, and RPAEF [4]. However, unlike these schemes, SonicAE offers the stronger dae security guarantee. For larger values of t and/or k, SonicAE's processing rate approaches $2n$ bits per F call, which is significantly higher and better than the rates of SotA AE schemes such as ZAE and Deoxys-II [26].

We provide a concrete performance comparison of SonicAE with other SotA AE schemes (ZAE, Deoxys-I [26], Deoxys-II, OCB3 [27], SIV [38], and CCM [41]) in Fig. 7 and Table 3. For this comparison, all FCs are instantiated with ForkSkinny and all TBCs/BCs with SKINNY. We set the associated data length to 0 to provide a conservative estimate of speedup, as increasing AD length would further improve SonicAE's relative performance.

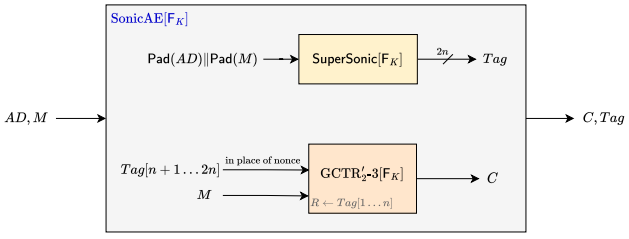


Fig. 6: The SonicAE AEAD mode. This figure illustrates the SIV-like composition processing injectively padded associated data AD and message M using the SuperSonic MAC mode and the GCTR'_2-3 encryption mode with key K. The second half of the $2n$-bit tag from SuperSonic is used as the effective nonce for GCTR'_2-3.

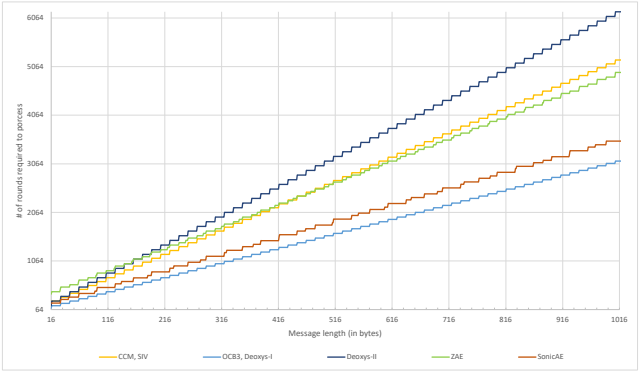


Fig. 7: Efficiency comparison of SonicAE (SIV composition of SuperSonic and GCTR'_2-3 instantiated with ForkSkinny) with other SotA AE modes (instantiated with SKINNY) for $n = k = 128$ bits. The plot shows the total number of primitive calls required to process a message of corresponding length (assuming 0 AD length).

As evidenced by Fig. 7 and Table 3, SonicAE offers significantly better performance than SotA AE schemes providing comparable nonce-misuse (mrae) security. Notably, SonicAE[ForkSkinny] achieves speed-ups of at least 1.41x (for 95B messages) and 1.32x (for 64KB messages) compared to ZAE[SKINNY] and Deoxys-II[SKINNY] under the setting $k = 128$, $t = n \in \{64, 128\}$.

Security of SonicAE. We formally state the claim regarding the dae security of SonicAE in Theorem 4. The detailed proof is deferred to App. C.

Theorem 4. *Let* F *be a forkcipher with* $\mathcal{T} = \{0,1\}^t$*. Then for any adversary* $\mathcal{A}$ *who makes at most* q *queries to* SonicAE *such that the total number of induced* F *calls is at most* σ*, we have*

$$\mathbf{Adv}^{\mathsf{dae}}_{\mathrm{SonicAE}[\mathsf{F}]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{xrtk\text{-}prtfp}}_{\mathsf{F}}(\mathcal{A}') + \frac{3q(\sigma+q)}{2^{n+\min\{n,t-2\}}} + \frac{q}{2^{2n}}$$

for some adversary $\mathcal{A}'$ *making at most* σ F *queries (under fixed but secret and random bases as per the* xrtk-prtfp *model), running in time bounded by the running time of* $\mathcal{A}$ *plus* $\gamma \cdot \sigma$*, where* γ *is the runtime of an* F *call.*

6 Conclusion

In this work, we introduced Sonikku, a novel family of fast and secure MACs based on tweakable expanding primitives, comprising the sequential BabySonic and DarkSonic, and the parallel SuperSonic. The Sonikku family provides significant advantages over state-of-the-art (T)BC-based MACs: substantially higher performance, beyond birthday to full n-bit security, smaller state, and design flexibility for diverse applications including resource-constrained IoTs.

BabySonic is particularly effective for short queries, offering strong security ($3n/4$ bits) and efficiency due to its lack of pre/post-processing and optimal

AE →	ZAE		Deoxys-II		OCB3, Deoxys-I		CCM, SIV	
Block size n (bits) →	64	128	64	128	64	128	64	128
IT nAE Security (bits) →	64	128	64	128	32, 64	64, 128	32	64
IT MRAE Security (bits) →	64	128	<64	<128	NIL	NIL	NIL, 32	NIL, 64
Max. message length	SonicAE speed-up/slow-down (multiplicative factor) 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

Table 3: Efficiency comparison of SonicAE (instantiated with ForkSkinny) against SotA AE modes (instantiated with SKINNY) with $k = 128$ bits, $n \in \{64, 128\}$ bits and $t \in \{0, n\}$ depending on the need of the compared AE mode. Entries show the ratio of the number of primitive calls required by the corresponding SotA AE mode relative to SonicAE for a given maximum message length (assuming 0 AD length). Green cells indicate where SonicAE is faster/better than the compared mode, while red indicates it is slower.

state size, achieving at least **2.11x** (and up to **4.36x**) speed-up over ZMAC ($n = t = k = 128$) for messages $\leq$ 95B.

DarkSonic improves upon BabySonic for longer messages, enhancing security (near n bits) and performance by using primarily one primitive output branch, providing nonce-misuse resistance with logarithmically degrading security upon nonce repetition.

SuperSonic offers message-length-independent security (near n bits) without a nonce, is fully parallelizable, and provides performance comparable to DarkSonic in sequential execution. DarkSonic and SuperSonic achieve speed-ups of at least **1.93x** and **1.48x**, respectively, against ZMAC ($k = 128, t = n \in \{64, 128\}$) for messages up to 95B and 64KB.

We also demonstrated the family's versatility by constructing SonicAE, an efficient, beyond-birthday secure, stateless, and deterministic authenticated encryption scheme based on SuperSonic.

Overall, Sonikku represents a significant advancement in designing fast, secure, and efficient MACs and AEADs leveraging expanding primitives, offering strong guarantees across various applications and environments.

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. Elena Andreeva was supported by the Austrian Science Fund (FWF) SpyCoDe grant with number 10.55776/F8507-N.

References

1. Andreeva, E., Bhati, A.S., Preneel, B., Vizár, D.: 1, 2, 3, Fork: Counter Mode Variants based on a Generalized Forkcipher. IACR Trans. Symmetric Cryptol. **2021**(3), 1–35 (2021)
2. Andreeva, E., Bhati, A.S., Vizár, D.: 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. pp. 512–534. Springer (2021)
3. Andreeva, E., Deprez, A., Pittevils, J., Roy, A., Bhati, A.S., Vizár, D.: New results and insighs on forkae. In: NIST LWC workshop (2020)
4. Andreeva, E., Lallemand, V., Purnal, A., Reyhanitabar, R., Roy, A., Vizár, D.: Forkcipher: a New Primitive for Authenticated Encryption of Very Short Messages. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 153–182. Springer (2019)
5. Andreeva, E., Weninger, A.: A forkcipher-based pseudo-random number generator. In: International Conference on Applied Cryptography and Network Security. pp. 3–31. Springer (2023)
6. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: CRYPTO 2016. pp. 123–153 (2016)
7. Bellare, M., Goldreich, O., Goldwasser, S.: Incremental Cryptography: The Case of Hashing and Signing. In: Desmedt, Y.G. (ed.) Advances in Cryptology — CRYPTO '94. pp. 216–233. Springer Berlin Heidelberg, Berlin, Heidelberg (1994)
8. Bellare, M., Kohno, T.: A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 491–506. Springer (2003)
9. Berti, F., Standaert, F.X., Levi, I.: Authenticity in the presence of leakage using a forkcipher. Cryptology ePrint Archive, Paper 2024/1325 (2024)
10. Bhargavan, K., Leurent, G.: On the Practical (In-)Security of 64-Bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN. In: ACM SIGSAC Conference on Computer and Communications Security. p. 456–467 (2016)
11. Bhati, A.S., Andreeva, E., Vizár, D.: OAE-RUP: a strong online AEAD security notion and its application to SAEF. In: International Conference on Security and Cryptography for Networks. pp. 117–139. Springer (2024)
12. Bhati, A.S., Dufka, A., Andreeva, E., Roy, A., Preneel, B.: Skye: An Expanding PRF based Fast KDF and its Applications. In: Proceedings of the 19th ACM Asia Conference on Computer and Communications Security. pp. 1082–1098 (2024)
13. Bhati, A.S., Pohle, E., Abidin, A., Andreeva, E., Preneel, B.: 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. pp. 2546–2560 (2023)
14. Bhati, A.S., Verbauwhede, M., Andreeva, E.: Breaking, Repairing and Enhancing XCBv2 into the Tweakable Enciphering Mode GEM. Cryptology ePrint Archive, Paper 2024/1554 (2024)
15. Black, J., Rogaway, P.: A Block-Cipher Mode of Operation for Parallelizable Message Authentication. In: EUROCRYPT 2002. pp. 384–397 (2002)
16. Cogliati, B., Lee, J., Seurin, Y.: New Constructions of MACs from (Tweakable) Block Ciphers. IACR Transactions on Symmetric Cryptology **2017**(2), 27–58 (2017)

17. Datta, N., Dutta, A., List, E., Mandal, S.: FEDT: Forkcipher-based leakage-resilient beyond-birthday-secure AE. IACR Communications in Cryptology (2024)
18. Datta, N., Dutta, A., Mancillas-López, C.: LightMAC: Fork it and make it faster. Advances in Mathematics of Communications **18**(5), 1406–1441 (2024)
19. Datta, N., Dutta, A., Nandi, M., Paul, G., Zhang, L.: Single key variant of PMAC_Plus. IACR Transactions on Symmetric Cryptology pp. 268–305 (2017)
20. Dong, X., Qin, L., Sun, S., Wang, X.: Key guessing strategies for linear key-schedule algorithms in rectangle attacks. In: Advances in Cryptology–EUROCRYPT 2022. pp. 3–33. Springer (2022)
21. Gaži, P., Pietrzak, K., Rybár, M.: The Exact PRF-Security of NMAC and HMAC. Cryptology ePrint Archive, Paper 2014/578 (2014)
22. Gunsing, A., Mennink, B.: The Summation-Truncation Hybrid: Reusing Discarded Bits for Free. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020. pp. 187–217. Springer International Publishing, Cham (2020)
23. Iwata, T., Kurosawa, K.: OMAC: One-Key CBC MAC. In: Johansson, T. (ed.) Fast Software Encryption. pp. 129–153. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
24. Iwata, T., Minematsu, K., Peyrin, T., Seurin, Y.: ZMAC: a fast tweakable block cipher mode for highly secure message authentication. In: Annual international cryptology conference. pp. 34–65. Springer (2017)
25. Jean, J., Nikolić, I., Peyrin, T.: Tweaks and keys for block ciphers: the TWEAKEY framework. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 274–288. Springer (2014)
26. Jean, J., Nikolić, I., Peyrin, T., Seurin, Y.: Deoxys v1.41 (2016), `https://competitions.cr.yp.to/round3/deoxysv141.pdf`
27. Krovetz, T., Rogaway, P.: The Software Performance of Authenticated-Encryption Modes. In: Joux, A. (ed.) Fast Software Encryption. pp. 306–327. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
28. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable Block Ciphers. In: CRYPTO 2002. pp. 31–46 (2002)
29. List, E., Nandi, M.: Revisiting full-PRF-secure PMAC and using it for beyond-birthday authenticated encryption. In: Cryptographers' Track at the RSA Conference. pp. 258–274. Springer (2017)
30. Liu, G., Ghosh, M., Song, L.: Security analysis of SKINNY under related-tweakey settings. Cryptology ePrint Archive (2016)
31. Luykx, A., Preneel, B., Tischhauser, E., Yasuda, K.: A MAC mode for lightweight block ciphers. In: International Conference on Fast Software Encryption. pp. 43–59. Springer (2016)
32. Naito, Y.: Full PRF-secure message authentication code based on tweakable block cipher. In: International Conference on Provable Security. pp. 167–182. Springer (2015)
33. Patarin, J.: The "Coefficients H" Technique. In: Selected Areas in Cryptography (SAC). p. 328–345 (2008)
34. Perrin, T.: The Noise protocol framework (2016), `noiseprotocol.org`
35. Purnal, A., Andreeva, E., Roy, A., Vizár, D.: What the Fork: Implementation Aspects of a Forkcipher. In: NIST Lightweight Cryptography Workshop 2019 (2019)
36. Qin, L., Dong, X., Wang, X., Jia, K., Liu, Y.: Automated Search Oriented to Key Recovery on Ciphers with Linear Key Schedule. IACR ToSC pp. 249–291 (2021)
37. Rogaway, P.: Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In: ASIACRYPT 2004. pp. 16–31 (2004)

38. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: EUROCRYPT 2006. pp. 373–390. Springer (2006)
39. Soliton: Teleoperation of Remote Machinery, https://www.solitonsystems.com/low-latency-video/remote-operation/remote-operation-of-machinery
40. TER: BRICK Radio Remote Control (2018), https://www.controldevices.group/PDFS/TER/TER%20Brick%20Radio%20Remote%20Control%20Data%20Sheet.pdf
41. Whiting, D., Housley, R., Ferguson, N.: Counter with CBC-MAC (CCM). IETF RFC 3610 (Informational) (Sep 2003), http://www.ietf.org/rfc/rfc3610.txt
42. Yasuda, K.: The sum of CBC MACs is a secure PRF. In: Cryptographers' Track at the RSA Conference. pp. 366–381. Springer (2010)
43. Yasuda, K.: A new variant of PMAC: beyond the birthday bound. In: Annual Cryptology Conference. pp. 596–609. Springer (2011)
44. YOURCONTROL: The Proxo Wireless Remote Control System, https://your-control.com/proxo/
45. Zhao, B., Dong, X., Meier, W., Jia, K., Wang, G.: 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 Security Analysis of **Sonikku** MACs

A.1 $\mathsf{BabySonic}_{n/a}$: Proof of Theorem 1

We prove the PRF security of $\mathsf{BabySonic}_{n/a}[\mathsf{F}]$ by bounding the advantage of any adversary $\mathcal{A}$ in distinguishing it from a uniform random function (URF). We use the standard indistinguishability game framework, comparing the real game (**PRF-real**) where $\mathcal{A}$ queries the MAC Π with a random secret key, against an ideal game (**PRF-ideal**) where the oracle returns a uniformly random string for each distinct query. The PRF advantage is $\mathbf{Adv}^{\mathsf{PRF}}_{\Pi}(\mathcal{A}) = \left|\Pr[\mathcal{A}^{\textbf{PRF-real}_{\Pi}} \Rightarrow 1] - \Pr[\mathcal{A}^{\textbf{PRF-ideal}_{\Pi}} \Rightarrow 1]\right|$. We slightly abuse the notation by dropping the bit 1 for brevity and denote it as $\left|\Pr[\mathcal{A}^{\textbf{PRF-real}_{\Pi}}] - \Pr[\mathcal{A}^{\textbf{PRF-ideal}_{\Pi}}]\right|$.

The $\mathsf{BabySonic}_{n/a}$ security proof utilizes a hybrid argument, replacing components of the real construction with ideal ones in a sequence of games. Let q be the number of queries made by $\mathcal{A}$, ℓ the maximum query length in blocks (i.e., $\ell = \max\{\ell^1, \ldots, \ell^q\}$ where ℓ^i is the length of the i^{th} query in $n + t + k$ bit blocks), and σ the total number of primitive calls across all q queries. We denote q_i as the number of queries with at least i primitive calls (i.e., containing at least i message blocks), such that $\sum_{i=1}^{\ell} q_i = \sigma$. Note that $q_1 = q$.

Recursively replacing F. We first replace the first $\mathsf{F}_{K_2 \oplus \mathsf{T}_2}(K_1 \oplus \mathsf{T}_1, \cdot)$ with a pair of independent random tweakable permutations $\pi_0 = (\pi_{\mathsf{T}_1 \| \mathsf{T}_2, 0} \leftarrow\!\!\$\ \mathrm{Perm}(n))_{\mathsf{T}_1 \| \mathsf{T}_2 \in \{0,1\}^{t+k}}$ and $\pi_1 = (\pi_{\mathsf{T}_1 \| \mathsf{T}_2, 1} \leftarrow\!\!\$\ \mathrm{Perm}(n))_{\mathsf{T}_1 \| \mathsf{T}_2 \in \{0,1\}^{t+k}}$ and let $\mathsf{BabySonic}'_{n/a}[\mathsf{STH2}[(\pi_0, \pi_1)], \mathsf{STH2}[\mathsf{F}], \ldots, \mathsf{STH2}[\mathsf{F}]]$ denote the $\mathsf{BabySonic}$ mode that uses (π_0, π_1) in place of the first F call, which yields $\mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{BabySonic}_{n/a}[\mathsf{F}]}(\mathcal{A}) \leq \mathbf{Adv}^{\textbf{xrtk-prtfp}}_{\mathsf{F}}(\mathcal{B}_{q_1}) + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{BabySonic}'_{n/a}[\mathsf{STH2}[(\pi_0,\pi_1)],\mathsf{STH2}[\mathsf{F}],\ldots,\mathsf{STH2}[\mathsf{F}]]}(\mathcal{A})$. Here $\mathcal{B}_{q_i}$ is an **xrtk-prtfp**-adversary against F that can make at most q_i queries to F.

We now replace the first (π_0, π_1) call and its following $\mathsf{STH2}$ call together with a random function $f = (f_{\mathsf{T}_1\|\mathsf{T}_2} \leftarrow\$ \mathrm{Func}(n, n+a))_{\mathsf{T}_1\|\mathsf{T}_2 \in \{0,1\}^{t+k}}$, denote it by $\mathsf{BabySonic}'_{n/a}[f, \mathsf{STH2[F]}, \ldots, \mathsf{STH2[F]}]$ and apply the summation-truncation results from [22, Theorem 2]. This gives us for $0 \le a \le \min\{n-10, 11n/12\}$,

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

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 $\mathsf{BabySonic}'_{n/a}[f, \mathsf{STH2[F]}, \ldots, \mathsf{STH2[F]}]$ 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 $10 \le a \le \min\{n-10, 11n/12\}$ and $q \le 2^{n-a/2}$,

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

Since $\sigma \ge q$, for $q > 2^{n-a/2}$ this bound becomes void and hence the assumption of $q \le 2^{n-a/2}$ can be dropped. After accounting for the advantages stemming from the primitive replacement and the STH composition, the remaining task is to bound the advantage of distinguishing the final hybrid game (where internal states are produced by random functions) from a true uniform random function i.e., distinguishing between the games $\mathsf{PRF\text{-}real}_{\mathsf{BabySonic}'_{n/a}[f,\ldots,f]}$ and $\mathsf{PRF\text{-}ideal}_{\mathsf{BabySonic}'_{n/a}[f,\ldots,f]}$. These games are indistinguishable if there are no collisions among the inputs of f calls that correspond to non-prefixed message blocks in $\mathsf{BabySonic}'_{n/a}[f, \ldots, f]$ over q queries. There are σ f calls in total, taking corresponding F's (X, T_1, T_2) triplet as input. $\sigma - q$ of these calls correspond to the message block processing steps (non-final calls), and q calls correspond to the final block processing. Since the internal states output by these random functions are independent, uniformly random $(n+a)$-bit values, we have $\mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{BabySonic}'_{n/a}[f,\ldots,f]}(\mathcal{A}) \le (\binom{\sigma-q}{2} + \binom{q}{2} - \binom{q}{2})/2^{n+a}$ 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). Thus, for $10 \le a \le \min\{n-10, 11n/12\}$, we have:

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

and hence the result of Theorem 1. □

A.2 SuperSonic: Proof of Theorem 2

We use the same definition of $\mathbf{Adv}^{\mathsf{PRF}}_{\Pi}(\mathcal{A})$ for a MAC Π as defined in App. A.1.

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

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

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 $P^i_1, \ldots, P^i_{\ell^i-1}$ (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 $(\Delta^i_1, \Delta^i_2, \Delta^i_3)$s where Δ^i_1, Δ^i_2 and Δ^i_3 correspond to the final chaining values used in (or XORed to) the final primitive call's input, tweak T_1 and tweak T_2, respectively. Let us denote the internally processed π_0 calls' outputs as α^i_js (corresponding to $P^i_j = M^i_{3j-2}\|M^i_{3j-1}\|M^i_{3j}$) then we can define Δs as $\Delta^i_1 = \oplus_{j=1}^{\ell^i-1} 2^{\ell^i-j}\alpha^i_j$, $\Delta^i_2 = \oplus_{j=1}^{\ell^i-1}(\alpha^i_j \oplus M_{3j-1})$ and $\Delta^i_3 = \oplus_{j=1}^{\ell^i-1} M_{3j}$. Let us now define the bad events when the real world can be distinguished from the ideal world.

BadT_1 a.k.a. "Final Input Collision in Real-world": There exists a pair of queries $1 \le i' < i \le q$ such that, the (i, ℓ^i) block call has tweak-input collision with the $(i', \ell^{i'})$ block call, i.e., $\mathsf{T}^i_{\ell^i} = \mathsf{T}^{i'}_{\ell^{i'}}$ and $\Delta^i_2 = \Delta^{i'}_2$.

BadT_2 a.k.a. "Output Collision in Ideal-world": There exists a pair of queries $1 \le i' < i \le q$ 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 $\mathsf{T}^i_{\ell^i} = \mathsf{T}^{i'}_{\ell^{i'}}$, $\Delta^i_2 \ne \Delta^{i'}_2$ and in ideal world we have $(Tag^i_1 = Tag^{i'}_1) \vee (Tag^i_2 = Tag^{i'}_2)$.

Note that the last condition in BadT_2 of ideal-world tag collision is independent of the other two real-world conditions. Let us define $\mathsf{BadT}_{2\mathsf{r}}$ (resp., $\mathsf{BadT}_{2\mathsf{i}}$) as BadT_2 with the ideal-world tag collision condition (resp., the other two real-world conditions) removed. Now, it is easy to see that for $\Pi_{\mathsf{ss}} = \mathsf{SuperSonic}[(\pi_0, \pi_1)]$ we have $\mathbf{Adv}^{\mathsf{PRF}}_{\Pi_{\mathsf{ss}}}(\mathcal{A})$

$$
\begin{aligned}
&\leq \Pr[\mathsf{BadT}_1] + \left|\Pr[\mathcal{A}^{\textbf{PRF-real}_{\Pi_{\mathsf{ss}}}} \wedge \neg\mathsf{BadT}_1] - \Pr[\mathcal{A}^{\textbf{PRF-ideal}_{\Pi_{\mathsf{ss}}}} \wedge \neg\mathsf{BadT}_1]\right| \\
&= \Pr[\mathsf{BadT}_1] + \left|\Pr[\mathcal{A}^{\textbf{PRF-real}_{\Pi_{\mathsf{ss}}}} \wedge \mathsf{BadT}_{2\mathsf{r}}] - \Pr[\mathcal{A}^{\textbf{PRF-ideal}_{\Pi_{\mathsf{ss}}}} \wedge \mathsf{BadT}_{2\mathsf{r}}]\right| \\
&= \Pr[\mathsf{BadT}_1] + \Pr[\mathsf{BadT}_{2\mathsf{r}}] \cdot \left|\Pr[\mathcal{A}^{\textbf{PRF-real}_{\Pi_{\mathsf{ss}}}} \mid \mathsf{BadT}_{2\mathsf{r}}] - \Pr[\mathcal{A}^{\textbf{PRF-ideal}_{\Pi_{\mathsf{ss}}}} \mid \mathsf{BadT}_{2\mathsf{r}}]\right| \\
&\leq \Pr[\mathsf{BadT}_1] + \Pr[\mathsf{BadT}_{2\mathsf{r}}] \cdot \Pr[\mathsf{BadT}_{2\mathsf{i}}] = \Pr[\mathsf{BadT}_1] + \Pr[\mathsf{BadT}_2]\,. \qquad (2)
\end{aligned}
$$

Distribution of Δs. 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 Δs 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 $\Pr[\Delta^i_1 = c_1 \wedge \Delta^i_2 = c_2] = \Pr[\Delta^i_1 = c_1] \cdot \Pr[\Delta^i_2 = c_2] \leq 1/(2^n - q)^2$.
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 $\Pr[\Delta^i_1 = c_1 \wedge \Delta^i_2 = c_2] = \Pr[\Delta^i_1 = c_1] \cdot \Pr[\Delta^i_2 = c_2 \mid \Delta^i_1 = c_1] \leq (1/(2^n - q)) \cdot \Pr[\Delta^i_2 = c_2 \mid \Delta^i_1 = c_1]$.

   Note that since query i and i' share all except one block input, lets say a^{th} one (P^i_a), we have that $\Delta^i_1 \oplus \Delta^{i'}_1 = 2^{\ell-a}(\Delta^i_2 \oplus \Delta^{i'}_2 \oplus M^i_{3a-1} \oplus M^{i'}_{3a-1})[1 \ldots n]$ which means if $c_1 = \Delta^{i'}_1$ and $c_2 = \Delta^{i'}_2$ then one of the following two always holds: 1. $\Delta^i_3 \neq \Delta^{i'}_3$ and $\Pr[\Delta^i_1 = c_1 \wedge \Delta^i_2 = c_2] \leq (1/(2^n - q)) \cdot 1 = 1/(2^n - q)$. and 2. $\Delta^i_3 = \Delta^{i'}_3$ and $\Pr[\Delta^i_1 = c_1 \wedge \Delta^i_2 = c_2] \leq (1/(2^n - q)) \cdot 0 = 0$. Or in other words, $\Pr[\Delta^i_1 = \Delta^{i'}_1 \wedge \Delta^i_2 = \Delta^{i'}_2 \wedge \Delta^i_3 = \Delta^{i'}_3] = 0$.

Hence, we can say that for distinct messages $\Pr[\Delta^i_1 = \Delta^{i'}_1 \wedge \Delta^i_2 = \Delta^{i'}_2 \wedge \Delta^i_3 = \Delta^{i'}_3] \leq 1/(2^n - q)^2$.

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 $\mathsf{T}^i_{\ell^i} = \mathsf{T}^{i'}_{\ell^{i'}}$ and $\Delta^i_2 = \Delta^{i'}_2$. Now, as analyzed above we have that for all $i' < i$, the two Δ masks (Δ^i_1, Δ^i_2) 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 $\Pr[\mathsf{BadT_1}] \leq \frac{2q^2}{2^{n+\min\{n,t-2\}}}$ 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 $\mathsf{T}^i_{\ell^i} = \mathsf{T}^{i'}_{\ell^{i'}}$, $\Delta^i_2 \neq \Delta^{i'}_2$ but $(Tag^i_1 = Tag^{i'}_1) \vee (Tag^i_2 = Tag^{i'}_2)$. 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 $\Pr[\mathsf{BadT}_2] \leq \frac{q^2}{2} \cdot \frac{2}{2^{\min\{n,t-2\}}} \left(\frac{1}{2^n} + \frac{1}{2^n}\right) = \frac{2q^2}{2^{n+\min\{n,t-2\}}}$ for $q \leq 2^{n-1}$.

Now, combining with Exp. 2, we get that $\mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{SuperSonic}[(\pi_0,\pi_1)]}(\mathcal{A}) \leq \frac{4 \cdot q^2}{2^{n+\min\{n,t-2\}}}$ (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. □

A.3 DarkSonic: Proof of Theorem 3

Coefficient H Technique. The coefficient H is a simple but powerful proof technique by Patarin [33]. 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 $\tau = \langle (M_1, T_1), \ldots, (M_q, T_q) \rangle$. The goal of an adversary $\mathcal{A}$ is to distinguish interactions in the real world $\mathcal{O}_{\mathsf{real}}$ from the ones in ideal world $\mathcal{O}_{\mathsf{ideal}}$.

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

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

Proof of Theorem 3. We use the same definition of $\mathbf{Adv}^{\mathsf{PRF}}_{\Pi}(\mathcal{A})$ for a MAC Π as defined in App. A.1.

Replacing F. We first replace $\mathsf{F}_{K_2 \oplus \mathsf{T}_2}(K_1 \oplus \mathsf{T}_1, \cdot)$ with a pair of independent random tweakable permutations $\pi_0 = (\pi_{\mathsf{T}_1 \| \mathsf{T}_2, 0} \leftarrow\$ \mathrm{Perm}(n))_{\mathsf{T}_1 \| \mathsf{T}_2 \in \{0,1\}^{t+k}}$ and

$\pi_1 = (\pi_{\mathsf{T}_1\|\mathsf{T}_2,1} \leftarrow\!\$\ \mathrm{Perm}(n))_{\mathsf{T}_1\|\mathsf{T}_2\in\{0,1\}^{t+k}}$ and let $\mathsf{DarkSonic}[(\pi_0,\pi_1)]$ denote the DarkSonic mode that uses π_0, π_1 instead of F, which yields $\mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{DarkSonic}[\mathsf{F}]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{xrtk\text{-}prtfp}}_{\mathsf{F}}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{DarkSonic}[(\pi_0,\pi_1)]}(\mathcal{A})$.

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

Transcripts. Following the coefficients H technique [33], we describe the interactions of $\mathcal{A}$ with its oracles in a *transcript*:

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

For the i^{th} query to the MAC oracle with input M^i (inc. nonce N^i as M^i_1) 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 $P^i_1, \ldots, P^i_{\ell^i-1}, P^i_*$ (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_0s 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 $\Delta^{i,\ell^i}\|\Delta^{i,\ell^i+1}$.

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 $j \leq \mathsf{llcp}_{n+k+t-2}(M^i, M^{i'})$ for some $i' < i$. To simplify the notations further, we let $\mathsf{llcp}_{n+k+t-2}(i)$ denote $\max_{1\leq i'<i} \mathsf{llcp}_{n+k+t-2}(M^i, M^{i'})$. Hence, a primitive query (i,j) will be considered as "prefixed-delta" if $j \leq \mathsf{llcp}_{n+k+t-2}(i)$.
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 $j = \mathsf{llcp}_{n+k+t-2}(i)+1$ and $(M^i \oplus M^{i'})[(j-1)(n+t+k-2)+n+1 \ldots (j-1)(n+t+k-2)+n+k+t-2] = 0^{k+t-2}$ 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

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

where $P^i_{\ell_i} = P^i_*$.

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:

$\mathsf{BadT_1}$ a.k.a. "Input Collision": There exists $(i',j') < (i,j)$ with $(j' > \mathsf{llcp}_{n+k+t-2}(i')) \wedge (j > \mathsf{llcp}_{n+k+t-2}(i))$, $1 \le j \le \ell^i, 1 \le j' \le \ell^{i'}$ 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 $(j \notin \{1,\ell^i\} \wedge j' \notin \{1,\ell^{i'}\}) \vee (j = \ell^i \wedge j' = \ell^{i'})$, $P^i_j \oplus P^{i'}_{j'} = (\Delta^{i,j-2} \oplus \Delta^{i',j'-2}) \| (\Delta^{i,j-1} \oplus \Delta^{i',j'-1})[1 \ldots z] \| 0^{k+t-2-z}$
2. or with $j = 1 \wedge j' \notin \{1,\ell^{i'}\}$, $P^i_j \oplus P^{i'}_{j'} = \Delta^{i',j'-2} \| \Delta^{i',j'-1}[1 \ldots z] \| 0^{k+t-2-z}$
3. or with $j \notin \{1,\ell^i\} \wedge j' = 1$, $P^i_j \oplus P^{i'}_{j'} = \Delta^{i,j-2} \| \Delta^{i,j-1}[1 \ldots z] \| 0^{k+t-2-z}$

$\mathsf{BadT_2}$ a.k.a. "Output Collision": There exists $(i',j') < (i,j)$ with $(j' > \mathsf{llcp}_{n+k+t-2}(i')) \wedge (j > \mathsf{llcp}_{n+k+t-2}(i))$, $1 \le j \le \ell^i, 1 \le j' \le \ell^{i'}$ and $\max\{j,j'\} \neq 1$ such that, the $(i,j)^{th}$ primitive call has tweak-output collisions with the $(i',j')^{th}$ primitive call given that the inputs to these calls are distinct, i.e. for $z = \min\{n, t-2\}$

1. with $(j \notin \{1,\ell^i\} \wedge j' \neq 1) \vee (j = \ell^i \wedge j' \notin \{1,\ell^{i'}\})$, $(P^i_j \oplus P^{i'}_{j'})[n+1 \ldots n+k+t-2] = (\Delta^{i,j-1} \oplus \Delta^{i',j'-1})[1 \ldots z] \| 0^{k+t-2-z}$, $(P^i_j \oplus P^{i'}_{j'})[1 \ldots n] \neq \Delta^{i,j-2} \oplus \Delta^{i',j'-2}$ and $\Delta^{i,j} = \Delta^{i',j'}$
3. or with $j \notin \{1,\ell^i\} \wedge j' = 1$, $(P^i_j \oplus P^{i'}_{j'})[n+1 \ldots n+k+t-2] = \Delta^{i,j-1}[1 \ldots z] \| 0^{k+t-2-z}$, $(P^i_j \oplus P^{i'}_{j'})[1 \ldots n] \neq \Delta^{i,j-2}$ and $\Delta^{i,j} = \Delta^{i',j'}$
3. or with $j = 1 \wedge j' \notin \{1,\ell^{i'}\}$, $(P^i_j \oplus P^{i'}_{j'})[n+1 \ldots n+k+t-2] = \Delta^{i',j'-1}[1 \ldots z] \| 0^{k+t-2-z}$, $(P^i_j \oplus P^{i'}_{j'})[1 \ldots n] \neq \Delta^{i',j'-2}$ and $\Delta^{i,j} = \Delta^{i',j'}$

4. or with $j = \ell^i \wedge j' = \ell^{i'}$, $(P^i_j \oplus P^{i'}_{j'})[n+1 \dots n+k+t-2] = (\Delta^{i,j-1} \oplus \Delta^{i',j'-1})[1 \dots z] \| 0^{k+t-2-z}$, $(P^i_j \oplus P^{i'}_{j'})[1 \dots n] \neq \Delta^{i,j-2} \oplus \Delta^{i',j'-2}$ 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 T_1'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 BadT_1 (or BadT_2, respectively). We denote by $\mathcal{T}_{\text{bad}}$, the set of "bad" transcripts that is defined as the subset of attainable transcripts for which the transcript predicate $\mathsf{BadT}(\tau) = (\mathsf{BadT}_1(\tau) \vee \mathsf{BadT}_2(\tau)) = 1$. We denote by $\mathcal{T}_{\text{good}}$, the set of attainable transcripts which are not in the set $\mathcal{T}_{\text{bad}}$.

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

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

Lemma 3. *Let $\tau \in \mathcal{T}_{good}$ 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 App. B.

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

B Omitted Lemma Proofs

B.1 Proof of Lemma 2

BadT_1. For any transcript in $\mathcal{T}_{\text{bad}}$ with BadT_1 set to 1, we know that there exists $(i', j') < (i, j)$ with $(j' > \mathsf{llcp}_{n+k+t-2}(i')) \wedge (j > \mathsf{llcp}_{n+k+t-2}(i))$, $1 \leq j \leq \ell^i, 1 \leq j' \leq \ell^{i'}$ and $\max\{j, j'\} \neq 1$ such that, one of the three statements of BadT_1 (as defined above) holds.

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

I. When $j = j' = \mathsf{llcp}_{n+k+t-2}(M^i, M^{i'}) + 1$: $\Delta^{i,j-1} = \Delta^{i',j'-1}$ and $\Delta^{i,j-2} = \Delta^{i',j'-2}$. However, the message parts $P^i_j \neq P^{i'}_{j'}$ and therefore, any of the targeted BadT_1 collisions can occur here with probability 0.

II. When $j = j' = \mathsf{llcp}_{n+k+t-2}(M^i, M^{i'}) + 2 > 2$: $\Delta^{i,j-2} = \Delta^{i',j'-2}$. 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 BadT_1 collisions can occur here with probability $\leq 2^{n-z}/(2^n - q)$.

III. Otherwise: all mentioned Δs here are relatively fresh which means in the ideal world, each one of these Δs 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 Δs are chosen using a random permutation or a random permutation pair without replacement). Hence, any of the targeted BadT_1 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 BadT_1. 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 $q \leq 2^{n-1}$, $\Pr[\mathsf{BadT}_1(\Theta_{id}) = 1] \leq \frac{\binom{\sigma-q}{2}+\binom{q}{2}-\binom{q}{2}-q(\mu-1)}{2^{n+z-2}} + \frac{q(\mu-1)}{2^{z-1}} \leq \frac{(\sigma-q)^2}{2^{n+z-1}} + \frac{q(\mu-1)}{2^{z-1}}$.

BadT_2. Similarly, for any transcript in $\mathcal{T}_{\text{bad}}$ with BadT_2 set to 1, we know that there exists $(i',j') < (i,j)$ with $(j' > \mathsf{llcp}_{n+k+t-2}(i')) \wedge (j > \mathsf{llcp}_{n+k+t-2}(i))$, $1 \leq j \leq \ell^i, 1 \leq j' \leq \ell^{i'}$ and $\max\{j,j'\} \neq 1$ such that, one of the four statements of BadT_2 (as defined above) holds.

Note that under any of these statements, we have the following cases for the mentioned Δs (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 $j = j' = \mathsf{llcp}_{n+k+t-2}(M^i, M^{i'}) + 1$: 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 Δs here are relatively fresh which means in the ideal world, each one of these Δs 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 $\binom{\sigma-q}{2}$ 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 the four statements. Out of these, $\binom{q}{2}$ pairs satisfy the above conditions I. Hence, we get under $q \leq 2^{n-1}$, $\Pr[\mathsf{BadT}_2(\Theta_{id}) = 1] \leq \frac{\binom{\sigma-q}{2}+\binom{q}{2}-\binom{q}{2}}{2^{n+z-3}} \leq \frac{2(\sigma-q)^2}{2^{n+z-1}}$.

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

B.2 Proof of Lemma 3

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 $\mathsf{T}_1\|\mathsf{T}_2$ 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 Δs within τ due to sharing a common prefix in their corresponding message with some previously queried message) over all $\sigma + 2q$ permutation calls. Let $y_{\mathsf{T}'}$ and $y'_{\mathsf{T}'}$ denote the number of permutation queries (i, j, b)s that share a same tweak $\mathsf{T}_1\|\mathsf{T}_2\|b$ as T' (for some $\mathsf{T}' \in \{0,1\}^{t+k+1}$) 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 $\mathcal{T}_{\mathsf{total}}$ and $\mathcal{T}_{\mathsf{first}}$, respectively. Clearly, with this, we can say $\sum_{\mathsf{T}'\in\mathcal{T}_{\mathsf{total}}} y_{\mathsf{T}'} = \sigma + 2q - x$ and $y_{\mathsf{T}'} \geq y'_{\mathsf{T}'}$.

Since in the ideal world, all these non-prefixed Δs 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 $g = \sum_{\mathsf{T}'\in\mathcal{T}_{\mathsf{first}}} y'_{\mathsf{T}'}$, $\Pr[\Theta_{id} = \tau] = (1^x \cdot (1/2^n)^{(\sigma+2q-x)-g} \cdot (\prod_{\mathsf{T}'\in\mathcal{T}_{\mathsf{first}}} 1/(2^n)_{y'_{\mathsf{T}'}}))$. On the other hand, in the real world these entities are computed using random tweaked permutations for similar queries which gives $\Pr[\Theta_{re} = \tau] = (1^x \cdot (\prod_{\mathsf{T}'\in\mathcal{T}_{\mathsf{total}}} 1/(2^n)_{y_{\mathsf{T}'}}))$ and consequently

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

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

C Proof of Theorem 4

Proof of Theorem 4. We provide the proof for the information-theoretic (IT) security of SonicAE$[(\pi_0, \pi_1)]$ where (π_0, π_1) is a pair of independent random tweakable permutations $\pi_0 = (\pi_{\mathsf{T}_1\|\mathsf{T}_2,0} \leftarrow\$ \mathrm{Perm}(n))_{\mathsf{T}_1\|\mathsf{T}_2\in\{0,1\}^{t+k}}$ and $\pi_1 =$

$(\pi_{\mathsf{T}_1\|\mathsf{T}_2,1} \leftarrow\$ \text{Perm}(n))_{\mathsf{T}_1\|\mathsf{T}_2\in\{0,1\}^{t+k}}$ replacing F in SonicAE (then from there, the proof for the computational counterpart is standard and straightforward).

The security of SIV-like constructions is bounded by the PRF security of the MAC component and the indistinguishability of the encryption component from a random function when used with a random nonce (derived from the MAC tag). Following the decomposition method for SIV schemes [38, Theorem 2], the dae advantage of SonicAE$[(\pi_0,\pi_1)]$ is bounded as:

$$\mathbf{Adv}^{\mathsf{dae}}_{\text{SonicAE}[(\pi_0,\pi_1)]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{SuperSonic}[(\pi_0,\pi_1)]}(\mathcal{A}') + \frac{q}{2^{2n}} + \mathbf{Adv}^{\mathsf{ive}}_{\text{GCTR}'_2\text{-3}[(\pi_0,\pi_1)]}(\mathcal{A}'') \tag{3}$$

for adversaries $\mathcal{A}'$ attacking SuperSonic's IT PRF security and $\mathcal{A}''$ attacking GCTR'_2-3's IT privacy (a nonce-based variant of IND-CPA security called ive [1]), both making at most q queries and inducing σ primitive calls. The term $q/2^{2n}$ accounts for the probability of collisions in the $2n$-bit tag output by SuperSonic over q queries.

The following result directly derived from [1, GCTR_2-3 proof and App. B.2] further bounds the ive advantage of GCTR'_2-3:

$$\begin{aligned}\mathbf{Adv}^{\mathsf{ive}}_{\text{GCTR}'_2\text{-3}[(\pi_0,\pi_1)]}(\mathcal{A}'') &\leq Pr[\mathcal{V} \text{ for } \text{GCTR}'_2\text{-3} \mid r = \min\{n, t-2\}] \\ &\quad + \frac{2(2\sigma - q)q}{2^{n+\min\{n,t-2\}+1}} \\ &\leq Pr[\mathcal{V} \text{ for } \text{GCTR}_2\text{-3} \mid x = q \wedge r = \min\{n, t-2\}] \cdot \max_{i\neq i'\leq q}\{\Pr[N^i = N^{i'}]\} \\ &\quad + \frac{(2\sigma - q)q}{2^{n+\min\{n,t-2\}}} \\ &\leq \frac{(2\sigma - q)q}{2^{\min\{n,t-2\}+1}} \cdot \frac{1}{2^n} + \frac{(2\sigma - q)q}{2^{n+\min\{n,t-2\}}} = \frac{6q\sigma - 3q^2}{2^{n+\min\{n,t-2\}+1}}\end{aligned} \tag{4}$$

where $\mathcal{V}$ is the event of cross-query input-tweak pair collisions. We refer the reader to [1] 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 $1/2^n$ 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. □

D Sonikku: Pseudocodes

```
function Pad(x, y, M)    // y ≤ x
    I'_pad ← 01; res ← |M|%x
    if res ≠ y then
        I'_pad ← 11
        if res < y then
            M ← M‖10^{y−1−res}
        else
            M ← M‖10^{x+y−1−res}
        end if
    end if
    return M, I'_pad
end function

function BabySonic(K, a, τ, M)    // k ≤ n + a
    K_1, K_2 ← Derive(K)
    M, I'_pad ← Pad(n + t + k − 2, 0, M)
    P_1, ..., P_{ℓ−1}, P_* ←(n+t+k−2)− M
    (Δ_1, Δ_2) ← (K_1, K_2)
    for i ← 1 to ℓ − 1 do
        M_{3i−2} ← P_i[1 ... n]
        M_{3i−1} ← P_i[n + 1 ... n + t − 2]
        M_{3i} ← P_i[n+t−1 ... n+t+k−2]
        T ← M_{3i−1}‖00
        (Δ'_1, Δ'_2) ← F^{Δ_2⊕T,b}_{Δ_1⊕M_{3i}}(M_{3i−2})
        Δ ← Δ'_1[1 ... a]‖Δ'_2[1 ... a]‖((Δ'_1 ⊕ Δ'_2)[a + 1 ... n])
        (Δ_1, Δ_2) ← (Δ[1 ... k], Δ[k + 1 ... n + a])
    end for
    T ← (P_*[n + 1 ... n + t − 2])‖I'_pad
    X, Y ← F^{Δ_2⊕T,b}_{Δ_1⊕P_*[n+t−1...n+t+k−2]}(P_*[1 ... n])
    Z ← X[1 ... a]‖Y[1 ... a]‖((X ⊕ Y)[a + 1 ... n])
    Tag ← Z[1 ... τ]
    return Tag
end function

function SuperSonic(K, e, τ, M)
    K_1, K_2 ← Derive(K)
    M, I'_pad ← Pad(n + t + k − e, 0, M)
    P_1, ..., P_{ℓ−1} ←(n+t+k−e)− M
    Δ_1, Δ_2, Δ_3 ← 0^n
    for i ← 1 to ℓ − 1 do
        M_{3i−2} ← P_i[1 ... n]
        M_{3i−1} ← P_i[n + 1 ... n + t − e]
        M_{3i} ← P_i[n+t−e+1 ... n+t+k−e]
        T ← M_{3i−1}‖⟨i⟩_{e−2}‖00
        Δ_1 ← (F^{K_2⊕T,0}_{K_1⊕M_{3i}}(M_{3i−2}) ⊕ Δ_1) ⊕_{t−2} M_{3i−1}
        Δ_2 ← 2 · (F^{K_2⊕T,0}_{K_1⊕M_{3i}}(M_{3i−2}) ⊕ Δ_2)
        Δ_3 ← M_{3i} ⊕ Δ_3
    end for
    T ← Δ_1‖I'_pad
    X, Y ← F^{K_2⊕T,b}_{K_1⊕Δ_3}(Δ_2)
    Tag ← (X‖Y)[1 ... τ]
    return Tag
end function

function DarkSonic(K, τ, N, M)
    K_1, K_2 ← Derive(K)
    M ← N‖M
    M, I'_pad ← Pad(n+t+k−2, t+k−2, M)
    P_1, ..., P_{ℓ−1}, P_* ←(n+t+k−2)− M
    (Δ_1, Δ_2) ← (0^n, 0^n)
    for i ← 1 to ℓ − 1 do
        M_{3i−2} ← P_i[1 ... n]
        M_{3i−1} ← P_i[n + 1 ... n + t − 2]
        M_{3i} ← P_i[n+t−1 ... n+t+k−2]
        T ← (M_{3i−1} ⊕_{t−2} Δ_2)‖00
        if i = 1 then
            (Δ_2, Δ_1) ← F^{K_2⊕T,b}_{K_1⊕M_{3i}}(M_{3i−2})
        else
            Δ ← Δ_2
            Δ_2 ← F^{K_2⊕T,0}_{K_1⊕M_{3i}}(M_{3i−2} ⊕ Δ_1)
            Δ_1 ← Δ
        end if
    end for
    T ← (P_*[1 ... t − 2] ⊕_{t−2} Δ_2)‖I'_pad
    X, Y ← F^{K_2⊕T,b}_{K_1⊕P_*[t−1...t+k−2]}(Δ_1)
    Tag ← (X‖Y)[1 ... τ]
    return Tag
end function
```

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