BUILDING THE NEXT GENERATION OF AUTHENTICATED ENCRYPTION

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Sanketh Gora Menda

August 2025

BUILDING THE NEXT GENERATION OF AUTHENTICATED ENCRYPTION

Sanketh Gora Menda, Ph.D.

Cornell University 2025

Symmetric encryption is the foundation for secure communication, and its current iteration in widespread use today is authenticated encryption with associated data (AEAD). But, the AEAD schemes in widespread use today like AES-GCM are beginning to show their age in the context of modern workloads whose scale and complexity defies assumptions made during their design over two decades ago. In the last few years, researchers and practitioners have uncovered a series of such critical limitations, and responded by proposing and deployed a patchwork of fixes addressing each of these limitations in isolation. This has led to a zoo of incompatible schemes with different security properties, which presents a challenge for analysis and interoperability.

This thesis argues for simplifying this cluttered landscape of AEAD schemes by building a new generation of clean-slate AEAD schemes targeting modern workloads. First, we emphasize the need for new schemes by introducing new attacks. We demonstrate the first commitment attacks against CCM, EAX, and SIV, and provide more versatile attacks against GCM and OCB3. Then, we specify the first of these new schemes: a new general-purpose AEAD scheme called OCH. It is the first scheme to simultaneously achieve 128-bit multi-user AE security, 128-bit context commitment security, and 256-bit nonces with optional nonce privacy. Finally, we consider the ever-incrasing list of special cases that do not admit a general-purpose AEAD scheme like OCH. Rather than specifying and analyzing a new scheme for every special case, we propose a new type of AEAD that flexibly incorporates multiple requirements simplifying analysis and usage.

అంకితం అమ్మ నాన్న కి

To Tom, for believing in me even when I didn’t

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Authenticated encryption with associated data (AEAD) underlies the security of virtually all secure systems including secure communication protocols, end-to-end encrypted messaging, and disk encryption. It works as follows [144]: the sender and the recipient agree on an AEAD scheme AEAD and a secret key K. To send a message M, the sender picks a nonce N unique to this message (like the packet number or a random string) and some associated data A it wants to authenticate but not hide (like the protocol version or the packet headers). Then, the sender feeds these values to the encryption function $\mathsf{AEAD.Enc}(K, N, A, M)$ to produce a ciphertext C which is sent to the recipient via an untrusted channel. On receipt of this ciphertext C and with knowledge of the key K, nonce N, and associated data A, the recipient runs the decryption function $\mathsf{AEAD.Dec}(K, N, A, C)$ to recover the message M, or receive an error if the ciphertext is malformed. Often, the nonce and associated data are transmitted along with the ciphertext over the untrusted channel.

The classic security goals in this setting are confidentiality and authenticity [144], which requires that adversaries without knowledge of the key K cannot learn or surreptitiously tamper with the enclosed message. These are the goals targeted by widely used AEAD schemes like AES-GCM [124].

But in the last few years, it has become increasingly clear that these widely used AEAD schemes like AES-GCM [124] suffer from many critical limitations in the context of modern workloads. These limitations can be split into two classes: deployment challenges and upgraded security goals. First, modern systems vary significantly from the networks of 2004 that GCM was designed for [124]. On the one end we have distributed workloads that encrypt can 2^{32} messages in a couple seconds [107], and on the other end we have tightly constrained but network-connected lightweight devices [162]. Second, modern systems introduce new attack vectors that lie outside the classic security goals.

For example, GCM allows adversaries to craft ciphertexts that can be decrypted by under multiple maliciously chosen keys, and this property has led to a series of serious vulnerabilities [118, 2, 125, 71, 86, 157, 100].

Against this backdrop, academics and practitioners have proposed and deployed a variety of fixes. Some are generic transformations [22, 2, 15, 52] without concrete realizations, others sound simple like prepending zeros to the message [2], yet others are tailored to specific current schemes like GCM to minimize modifications [15]. The only constant among these fixes is that they only address a strict subset of the critical limitations of current schemes. This has led to a plethora of incompatible schemes with different security properties.

This thesis argues for replacing the aging schemes in widespread use today with a new generation of clean-slate AEAD schemes that simultaneously address all the limitations of the current schemes, and are organized to be easy-to-use. Taken together, the next three chapters provide a blueprint for realizing this dream. We now summarize the chapters in turn.

In Chapter 2, we resolve open questions about the commitment security of standardized and deployed schemes CCM, EAX, and SIV; introduce a new class of commitment attacks against GCM, OCB3, CCM, EAX, and SIV; and introduce a new granular framework for commitment security that encompasses these attacks and derive relationships between prior attacks and the new attacks. This chapter is based on [125, 31, 126].

In Chapter 3, we specify OCH, the first AEAD scheme designed to simultaneously addresses pressing limitations of currently widely-deployed schemes. We construct and formally analyze the security of OCH in a modular fashion, with transforms that are of broader applicability. On Intel Raptor Lake CPUs, OCH using the Areion permutation family has a peak encryption speed of 0.62 cycles per byte (cpb), not far off from AES128-GCM (0.38cpb) and outperforming both ChaCha20/Poly1305 (1.63cpb) and TurboSHAKE128-Wrap (3.52cpb). This chapter is based on [128, 32].

In Chapter 4, we define and build a new type of AEAD scheme that we call flexible. Flexibility is intended as an answer to the growing list of desired security and performance features for future AEAD standards. Rather than a scheme per requirement, we offer a single scheme that flexibly incorporates multiple requirements, yet in a unified, systematic, and performance-optimal way. In addition, the underlying construction is configurable through an application-chosen input called a configuration. This configuration lists desired security and performance attributes like nonce-misuse resistance, robustness, parallelizability, and side-channel resistant software implementations. These attributes can be modified dynamically, and the scheme will adapt and provide the chosen set of attributes without requiring expensive key rotation. To build a flexible scheme, we take a clean-slate approach. Our proposal Flex is built modularly from a single permutation, simplifying analysis and implementation. This chapter is based on [127, 32].

CHAPTER 2

CONTEXT DISCOVERY AND COMMITMENT ATTACKS*

Designers of authenticated encryption with associated data (AEAD) have traditionally targeted security in the sense of confidentiality and ciphertext integrity, first in the context of randomized authenticated encryption [19], and then nonce-based [145] and misuse-resistant AEAD [148].

But in recent years researchers and practitioners have begun realizing that confidentiality and integrity as previously formalized prove insufficient in a variety of contexts. In particular, the community is beginning to appreciate the danger of schemes that are not key committing, meaning that an attacker can compute a ciphertext such that it can successfully decrypt under two (or more) keys. Non-key-committing AEAD was first shown to be a problem in the context of moderation in encrypted messaging [71, 87], and later in password-based encryption [119], password-based key exchange [119], key rotation schemes [3], and symmetric hybrid (or envelope) encryption [3].

Even more recently, new definitions have been proposed [16] that target committing to the key, associated data, and nonce. And while there have been proposals for new schemes [3, 16] that meet these varying definitions, questions still remain about which current AEAD schemes are committing and in which ways. Moreover, there have been no commitment results shown for a number of important practical AEAD schemes, such as CCM [74], EAX [28], and SIV [148]. Implementing (and standardizing) new AEAD schemes takes time and so understanding which standard AEAD schemes can be securely used in which settings is a pressing issue.

*This chapter is joint work with Julia Len, Paul Grubbs, and Thomas Ristenpart. The proceedings version of this chapter appeared at Eurocrypt 2023 [125], and an earlier version was presented at Real World Crypto 2023 [31]. This chapter is lightly edited from full version [126].

This work makes four main contributions. First, we provide a new, more granular framework for commitment security, which expands on prior ones to better capture practical attack settings. Second, we show the first key commitment attack against the original SIV mode, which was previously an open question. Third, we introduce a new kind of commitment security notion for AEAD—what we call *context discoverability*—which is analogous to preimage resistance for cryptographic hash functions. Fourth, we give context discovery attacks against a range of schemes which, by a general implication, also yield new commitment attacks against those schemes. A summary of our new attacks, including comparison with prior ones, when relevant, is given in Figure 2.1.

Granular commitment notions. Recall that a nonce-based AEAD encryption algorithm Enc takes as input a key K, nonce N, associated data A, and a message M. It outputs a ciphertext C. Decryption Dec likewise takes in a (K, N, A) triple, which we call the decryption *context*, along with a ciphertext C, and outputs either a message M or special error symbol $\perp$.

While most prior work has focused on key commitment security, which requires commitment to only one part (the key) of the decryption context, Bellare and Hoang (BH) [16] suggest a more expansive sequence of commitment notions for nonce-based AEAD. For the first, CMT-1, an adversary wins if it efficiently computes a ciphertext C and two decryption contexts (K_1, N_1, A_1) and (K_2, N_2, A_2) such that decryption of C under either context works (does not output $\perp$) and $K_1 \neq K_2$. CMT-1 is often called key commitment.[1] CMT-3 relaxes the latter winning condition to allow a win should the decryption contexts differ in any way. We therefore refer to CMT-3 as *context commitment* and schemes that meet CMT-3 as *context committing*. These notions form

[1]BH refer to this as CMTD-1, but for tidy AEAD schemes, CMT-1 and CMTD-1 are equivalent, so we prefer the compact term.

a strict hierarchy, with CMT-3 being the strongest. Despite this, most prior attacks [71, 87, 119, 3] have focused solely on key commitment (CMT-1).

Our first contribution is to refine further the definitional landscape for nonce-based AEAD schemes in a way that is particularly useful for exploring context commitment attacks. In practice, attackers will often face application-specific restrictions preventing full control over the decryption context. For example, in the Dodis, Grubbs, Ristenpart, and Woodage (DGRW) [71] attacks against Facebook's message franking scheme, the adversary had to build a ciphertext that decrypts under two contexts with equivalent nonces. Their (in BH's terminology) CMT-1 attack takes on a special form, and we would like to be able to formally distinguish between attacks that achieve additional adversarial goals (e.g., different keys but equivalent nonces) and those that may not.

We therefore introduce a new, parameterized security notion that generalizes the BH notions. Our CMT[Σ] notion specifies what we call a setting $\Sigma = (\mathsf{ts}, \mathsf{S}, \mathsf{P})$ that includes a target specifier ts, a context selector S, and a predicate P. The parameters ts and S specify which parts of the context are attacker-controlled versus chosen by the game, and which of the latter are revealed to the attacker. Furthermore, the predicate P takes as input the two decryption contexts and decrypted messages, and outputs whether the pair of tuples satisfy a winning condition. An adversary wins if it outputs a ciphertext and two contexts satisfying the condition that each decrypt the ciphertext without error. The resulting family of commitment notions includes both CMT-1 and CMT-3 but also covers a landscape of further notions.

We highlight two sets of notions. The first set is composed of $\mathrm{CMT}_{\mathsf{k}}$, $\mathrm{CMT}_{\mathsf{n}}$, and $\mathrm{CMT}_{\mathsf{a}}$, which use predicates $(K_1 \neq K_2)$, $(N_1 \neq N_2)$, and $(A_1 \neq A_2)$, respectively. The first notion is equivalent to CMT-1; the latter two are new. All of them are orthogonal to each other and a scheme that meets all three simultaneously achieves CMT-3. We say

Scheme	**CDY$^*_\mathsf{a}$**	**CDY$^*_\mathsf{n}$**	**CMT$^*_\mathsf{a}$**	**CMT$^*_\mathsf{k}$**	**CMT$_\mathsf{k}$**	**CMT-3**
GCM [73]	★✗ §2.3	★✗ §2.3	★✗ §E	☆✗ [87, 71]	☆✗ [87, 71]	☆✗ [87, 71]
SIV [150]	★✗ §2.3			★✗ §2.4	★✗ ⏩	★✗ ⏩
CCM [74]	★✗ §2.3				★✗ ⏩	★✗ ⏩
EAX [28]	★✗ §2.3	★✗ §2.3			★✗ ⏩	★✗ ⏩
OCB3 [113]	★✗ §2.3			☆✗ [3]	☆✗ [3]	☆✗ [3]
PaddingZeros	★✓ ⏩	★✓ ⏩	★✗ §E	☆✓ [3]	☆✓ [16]	★✗ ⏩
KeyHashing	★✓ ⏩	★✓ ⏩	★✗ §E	☆✓ [3]	☆✓ [3]	★✗ ⏩
CAU-C1 [16]	★✓ ⏩	★✓ ⏩		☆✓ [16]	☆✓ [16]	★✗ ⏩

Figure 2.1: Summary of context discovery and commitment attacks against a variety of popular AEAD schemes. Symbol ✓ indicates a proof that any attack will take at least 2^{64} time, while symbol ✗ indicates the existence of an attack that takes less than 2^{64} time; symbol ★ indicates results new to this paper and ☆ indicates prior work (citation given). CMT$_\mathsf{k}$ and CMT-3 are from Bellare and Hoang [16], where CMT$_\mathsf{k}$ was called CMT-1. The notions CDY$^*_\mathsf{a}$, CDY$^*_\mathsf{n}$, CMT$^*_\mathsf{a}$, and CMT$^*_\mathsf{k}$ are introduced in this paper, and CDY$^*_\mathsf{a}$, CDY$^*_\mathsf{n}$, and CMT$^*_\mathsf{k}$ are implied by CMT$_\mathsf{k}$. Symbol ⏩ indicates that the result is implied from one of the other columns by a reduction shown in this paper. §E indicates Appendix E in the full version [126].

these notions are *permissive* because the predicates used do not make any demands on other components of the context. In contrast, *restrictive* variants, which we denote via CMT$^*_\mathsf{k}$, CMT$^*_\mathsf{n}$, and CMT$^*_\mathsf{a}$, require equality for other context components. For example, the first uses the predicate $(K_1 \neq K_2) \wedge ((N_1, A_1) = (N_2, A_2))$. These capture the types of restrictions faced in real attacks mentioned above.

Breaking the original SIV. While prior work has shown (in our terminology) CMT$^*_\mathsf{k}$ attacks for GCM [87, 71], GCM-SIV [157, 119], ChaCha20/Poly1305 [87, 119], XChaCha20/Poly1305 [119], and OCB3 [3], an open question of practical interest [158] is whether there also exists a CMT$^*_\mathsf{k}$ attack against Synthetic IV (SIV) mode [148]. We resolve this open question, showing an attack that works in time about 2^{53}. It requires new techniques compared to prior attacks.

SIV combines a PRF F with CTR mode encryption, encrypting by first computing a tag $T = F_K(N, A, M)$ and then applying CTR mode encryption to M, using T as the (synthetic) IV and a second key K'. The tag and CTR mode output are, together, the ciphertext. Decryption recovers the message and then recomputes the tag, rejecting the ciphertext if it does not match. Schmieg [157] and Len, Grubbs, and Ristenpart (LGR) [119] showed that when F is a universal hash-based PRF, in particular GHASH for AES-GCM-SIV, one can achieve a fast $\mathrm{CMT}_{\mathsf{k}}^{*}$ attack.

Their attack does not extend to other versions of SIV, perhaps most notably the original version that uses for F the S2V[CMAC] PRF [148]. This version has been standardized [94] and is available in popular libraries like Tink [4]. For brevity here we describe the simpler case where F is just CMAC; the body will expand on the details. At first it might seem that CMAC's well-known lack of collision resistance (for adversarially-chosen keys), should extend to allow a simple $\mathrm{CMT}_{\mathsf{k}}^{*}$ attack: find K_1, K_2 such that $T = \mathsf{CMAC}_{K_1}(N, A, M) = \mathsf{CMAC}_{K_2}(N, A, M')$ for $M \neq M'$. But the problem is that we need M, M' to also satisfy

$$M \oplus \mathsf{CTR}_{K_1'}(T) = M' \oplus \mathsf{CTR}_{K_2'}(T) \tag{2.1}$$

where $\mathsf{CTR}_K(T)$ denotes running counter mode with initialization vector T and block cipher key K. When using a GHASH-based PRF, the second condition "plays well" with the algebraic structure of the first condition, making it computationally easy to satisfy both simultaneously. But, here that does not work.

The core enabler for our attack is that we can recast the primary collision finding goal as a generalized birthday bound attack. For block-aligned messages, we show how the two constraints above can be rewritten as a single equation that is the xor-sum of four terms, each taking values over $\{0, 1\}^n$. Were the terms independently and uniformly random, one would immediately have an instance of a 4-sum problem, which can be solved using Wagner's k-tree algorithm [164] in time $\mathcal{O}(2^{n/3})$. But our terms are neither independent nor uniformly random. Nevertheless, our main technical lemma shows that, in the ideal cipher model, the underlying block cipher and the structure of the terms (which are dictated by the details of CMAC-SIV) allows us to analyze the distribution of these terms and show that we can still apply the k-tree algorithm and achieve the same running time. This technique may be of independent interest.

Using this, we construct a $\mathrm{CMT}_{\mathsf{k}}^{*}$ attack against S2V[CMAC]-SIV that works in time about 2^{53}, making it practical and sufficiently damaging to rule out SIV as suitable for contexts where key commitment matters.

Context discoverability. Next we introduce a new type of security notion for AEAD. The cryptographic hashing community has long realized the significance of definitions for both collision resistance and preimage resistance [41], the latter of which, roughly speaking, refers to the ability of an attacker to find some input that maps to a target output. In analyzing $\mathrm{CMT}_{\mathsf{k}}$ security for schemes, we realized that in many cases we can give very strong attacks that, given any ciphertext, can find a context that decrypts it—a sort of preimage attack against AEAD. To avoid confusion, we refer to this new security goal for AEAD as *context discoverability (CDY)*, as the adversary is tasked with efficiently computing ("discovering") a suitable context for some target ciphertext.

While we have not seen real attacks that exploit context discoverability, since CDY is to CMT what preimage resistance is to collision resistance, we believe that they are inevitable. We therefore view it beneficial to get ahead of the curve and analyze the CDY security before concrete attacks surface.

We formalize a family of CDY definitions similarly to our treatment for CMT. Our

$$\begin{tikzcd}
\mathrm{CMT}_{\mathsf{k}} \arrow[drr] \arrow[drrrr] & & \mathrm{CMT}_{\mathsf{n}} \arrow[dll] \arrow[drr] & & \mathrm{CMT}_{\mathsf{a}} \arrow[dllll] \arrow[dll] \\
\mathrm{CDY}^*_{\mathsf{k}} & & \mathrm{CDY}^*_{\mathsf{n}} & & \mathrm{CDY}^*_{\mathsf{a}}
\end{tikzcd}$$

$$\begin{tikzcd}
& & \text{CMT-3} \arrow[dll] \arrow[dl] \arrow[d] \arrow[drr, dotted, "Th\ 1"] & & & \\
\mathrm{CMT}_{\mathsf{k}} \arrow[d] & \mathrm{CMT}_{\mathsf{n}} \arrow[d] & \mathrm{CMT}_{\mathsf{a}} \arrow[d] & & \mathrm{CDY}^* \arrow[dl] \arrow[d] \arrow[dr] & \\
\mathrm{CMT}^*_{\mathsf{k}} & \mathrm{CMT}^*_{\mathsf{n}} & \mathrm{CMT}^*_{\mathsf{a}} & \mathrm{CDY}^*_{\mathsf{k}} & \mathrm{CDY}^*_{\mathsf{n}} & \mathrm{CDY}^*_{\mathsf{a}}
\end{tikzcd}$$

Figure 2.2: **(Top)** Selected relationships between permissive CMT notions and restrictive CDY notions. Solid arrows represent implications. **(Bottom)** Selected relationships between CMT-3 and the notions we introduce in this paper. Solid arrows represent implications. The dotted arrow from CMT-3 to CDY* holds assuming "context compression" as defined in Theorem 1.

CDY[Σ] notion is parameterized by a setting $\Sigma = (\mathsf{ts}, \mathsf{S})$ that specifies a target specifier ts and a context selector S. Like for CMT[Σ], ts and S specify the parts of the context that the attacker can choose and which parts are chosen by the game and either hidden or revealed to the attacker. Unlike CMT, however, the attacker is always given a target ciphertext and needs to only produce one valid decrypting context.

Similar to $\mathrm{CMT}^*_{\mathsf{k}}$, $\mathrm{CMT}^*_{\mathsf{n}}$, $\mathrm{CMT}^*_{\mathsf{a}}$, we define the notions $\mathrm{CDY}^*_{\mathsf{k}}$, $\mathrm{CDY}^*_{\mathsf{n}}$, $\mathrm{CDY}^*_{\mathsf{a}}$. The notion $\mathrm{CDY}^*_{\mathsf{k}}$ captures the setting where an adversary is given arbitrary ciphertext C, nonce N, and associated data A, and must produce a key K such that C decrypts under (K, N, A). Similarly, $\mathrm{CDY}^*_{\mathsf{n}}$ and $\mathrm{CDY}^*_{\mathsf{a}}$ require the adversary to provide a nonce and associated data, respectively, given the other components chosen arbitrarily. These model restricted attack settings where parts of the context are not in the adversary's control.

We also define CDY*[ts] which generalizes this intuition to any target specifier ts. For example, in CDY*[$\mathsf{ts} = \{\mathsf{n}\}$] the adversary is given arbitrary ciphertext and nonce N, and must produce a key K and associated data A such that the ciphertext decrypts under (K, N, A).

We next analyze the relations between these sets of notions. In particular, we show that if an AEAD scheme is "context compressing"—ciphertexts are decryptable under more than one context—then CMT-3 security implies CDY*. This is analogous to collision resistance implying preimage resistance, though the details are different. Further, we observe that almost all deployed AEAD schemes are context compressing since they "compress" the nonce and associated data into a shorter tag. This allows us to focus on finding CDY$^*[\Sigma]$ attacks for AEAD schemes to show that these schemes also do not meet CMT$[\Sigma]$ security. Selected relationships are shown in Figure 2.2.

This opens up a new landscape of analysis, which we explore. We characterize a large class of AEAD schemes that use non-preimage resistant MACs and, based on this weakness, develop fast CDY$^*_{\mathsf{a}}$ attacks. The set includes CCM, EAX, SIV, GCM, and OCB3. For EAX and CCM, this represents the first attacks of any kind for committing security. For EAX and GCM, we are also able to give CDY$^*_{\mathsf{n}}$ attacks, which is perhaps even more surprising a priori, given that an adversary in this case only controls the nonce.

All this sheds light on the deficiencies of several popular design paradigms for AEAD, when viewed from the perspective of context commitment security. These definitions also allow us to precisely communicate attacks and threat models. For example, CDY might suffice for some applications while others might want the more computationally expensive CMT security.

Revisiting commitment-enhancing mechanisms Finally, in Appendix E in the full version [126] we use this new framework to analyze proposed mechanisms for commitment security. First, we look at the folklore padding zeros transform which prefixes zeroes to a message before encrypting and verifies the existence of these zeroes at decryption. This transform was recommended in an early OPAQUE draft specification [111, §3.1.1] and was shown by Albertini et al. [3, §5.3] to achieve FROB security and by Bellare and Hoang [16] to achieve CMT-1 security. We show that this transform does not achieve our $\text{CMT}^*_{\mathsf{a}}$ notion (and thus CMT-3) for all AEAD schemes, ruling it out as a candidate commitment security transform. We then make similar observations about the $\mathsf{CommitKey}$ transform which appends to the ciphertext a hash commitment to the key and the nonce. Finally, we conclude by considering the practical key commitment security of the recent CAU-C1 scheme from BH [16]. While a naive adaptation of DGRW's [71] "invisible salamanders" attack to this scheme takes about 2^{81} time, we show a more optimized attack which takes a little more than 2^{64} time, showing that 64-bit key-committing security does not preclude practical attacks.

Next steps and open problems. Our results resolve a number of open problems about AEAD commitment security, and overall highlight the value of new definitional frameworks that surface different avenues for attack. That said, we leave several open problems, such as whether different flavors of context discovery or commitment attacks can be found against popular schemes—the blank entries in Figure 2.1. Our attack techniques do not seem to work against these schemes, but whether positive security results can be shown is unclear.

2.1 Background

Notation. We refer to elements of $\{0,1\}^*$ as *bitstrings*, denote the length of a bitstring x by $|x|$ and the left-most (i.e., "most-significant") bit by $\mathsf{msb}(x)$. Given two bitstrings x and y, we denote their concatenation by $x \parallel y$, their bitwise xor by $x \oplus y$, and their bitwise and by $x \& y$. Given a number n, we denote its m-bit encoding as $\mathrm{encode}_m(n)$. For a finite set X, we use $x \leftarrow_{\$} X$ to denote sampling a uniform, random element from X and assigning it to x.

Sometimes, we operate in the finite field GF(2^n) with 2^n elements. This field is defined using an irreducible polynomial $f(\alpha)$ in GF(2)[α] of degree n.The elements of the field are polynomials $x_0 + x_1\alpha + x_2\alpha^2 + \cdots + x_{n-1}\alpha^{n-1}$ of degree $n-1$ with binary coefficients $x_i \in \mathrm{GF}(2)$. These polynomials can be represented by the n-bit string $x_0 x_1 \cdots x_{n-1}$ of their coefficients. Both addition and subtraction of two n-bit strings, denoted $x + y$ and $x - y$, respectively, is their bitwise xor $x \oplus y$. Multiplication of two n-bit strings, denoted $x \cdot y$, corresponds to the multiplication of the corresponding polynomials x and y followed by modular reduction with the irreducible polynomial $f(\alpha)$.

Probability. An *n-bit random variable* X is one whose value is probabilistically assigned, defined by *probability mass function* $p_X(x) := \Pr[X = x]$.The *n-bit uniform random variable* U is the random variable with the probability mass function $p_U(x) = \frac{1}{2^n}$ for all $x \in \{0,1\}^n$. Given two n-bit random variables X and Y, we define the *total variation distance* between them

$$\Delta(X, Y) := \max_{i \in \{0,1\}^n} \big| \Pr(X = i) - \Pr(Y = i) \big| .$$

A *random function* F from n-bit strings to m-bit strings is a collection $\{X_i : i \in \{0,1\}^n\}$ of m-bit random variables X_i, one for each n-bit input, such that for all $i \in \{0,1\}^n$, $F(i) := X_i$. A random function F from n-bit strings to m-bit strings is *uniformly random* if, for all $i \in \{0,1\}^n$, $F(i)$ is the m-bit uniform random variable.We say that two random functions F_1 and F_2 from n-bit strings to m-bit strings are *independent* if, for all $i \in \{0,1\}^n$ and for all $j \in \{0,1\}^n$, $F_1(i)$ and $F_2(j)$ are independent m-bit random variables.

Code-based games To formalize security experiments, we use the *code-based games* framework of Bellare and Rogaway [26]; with refinements from Ristenpart, Shacham, and Shrimpton [143]. A *procedure* P is a sequence of code-like statements that accepts some input and produces some output.We use superscripts like P^Q to denote that procedure P calls procedure Q.We use $(G \Rightarrow x)$ to denote the event that the procedure G outputs x, over the random coins of the procedure. Finally, given a game G and an adversary $\mathcal{A}$, we denote the *advantage* of $\mathcal{A}$ at G by $\mathbf{Adv}_G(\mathcal{A}) := \Pr[G(\mathcal{A}) \Rightarrow \mathsf{true}]$.

Cost of attacks. We represent cryptanalytic attacks by procedures and compute their cost using a *unit-cost RAM model.* Specifically, following [143], we use the convention that each pseudocode statement of a procedure runs in unit time. This lets us write the running time of a procedure as the maximum number of statements executed, with the maximum taken over all inputs of a given size. Similarly, we define the number of queries as the maximum number of queries executed over inputs of a given size. We recognize that this is a simplification of the real-world (e.g., see Wiener [166]), but for the attacks discussed in this paper, we nevertheless believe that it provides a good estimate.

Pseudorandom functions. A *pseudorandom function (PRF)* is a function $\mathsf{F} : \mathcal{K} \times \mathcal{M} \to \mathcal{Y}$ defined over a key space $\mathcal{K} \subseteq \{0,1\}^*$, message space $\mathcal{M} \subseteq \{0,1\}^*$, and output space $\mathcal{Y} \subseteq \{0,1\}^*$, that is indistinguishable from a uniform random function. More formally, we define the PRF advantage of an adversary $\mathcal{A}$ as

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A}) := \left|\Pr[K \leftarrow_{\$} \mathcal{K} : \mathcal{A}(\mathsf{F}(K,\cdot))] - \Pr[R \leftarrow_{\$} \mathrm{Func} : \mathcal{A}(R)]\right|,$$

and say that F is a PRF if this advantage is small for all adversaries $\mathcal{A}$ that run in a feasible amount of time.

Hash functions. A *hash function* is a function $\mathsf{H} : \mathcal{K} \times \mathcal{M} \to \mathcal{Y}$, defined over a key space $\mathcal{K} \subseteq \{0,1\}^*$, message space $\mathcal{M} \subseteq \{0,1\}^*$, and hash space $\mathcal{Y} \subseteq \{0,1\}^*$.We define the collision-resistance advantage of adversary $\mathcal{A}$ for H as

$$\mathbf{Adv}^{\mathrm{coll}}_{\mathsf{H}}(\mathcal{A}) := \Pr\left[K \leftarrow_{\$} \mathcal{K}, (M_1, M_2) \leftarrow_{\$} \mathcal{A}(K) : (M_1 \neq M_2) \text{ and } (\mathsf{H}(K, M_1) = \mathsf{H}(K, M_2))\right].$$

Block ciphers and the ideal cipher model. An n-bit *block cipher*, or a block cipher with *block length n* bits, is a function $E : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$, where for each key $k \in \{0,1\}^n$, $E(k, \cdot)$ is a permutation on $\{0,1\}^n$. Since it is a permutation, it has an inverse which we denote by $E^{-1}(k, \cdot)$. To simplify notation, we sometimes use the shorthands $E_k(\cdot) := E(k, \cdot)$ and $E_k^{-1}(\cdot) := E^{-1}(k, \cdot)$.

An n-bit *ideal block cipher* [108] is a random map $E : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$, such that for each key $k \in \{0,1\}^n$, $E_k(\cdot)$ is a permutation on $\{0,1\}^n$. Alternatively, we can think of an ideal block cipher as one where for each key $k \in \{0,1\}^n$, $E_k(\cdot)$ is uniformly, randomly sampled from the set of permutations on n-bits.

Authenticated encryption schemes. An *AEAD scheme* is a triple of algorithms $\mathsf{AEAD} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$, defined over a key space $\mathcal{K} \subseteq \{0,1\}^*$, nonce space $\mathcal{N} \subseteq \{0,1\}^*$, associated data space $\mathcal{A} \subseteq \{0,1\}^*$, message space $\mathcal{M} \subseteq \{0,1\}^*$, and ciphertext space $\mathcal{C} \subseteq \{0,1\}^*$.

1. $\mathsf{Kg} : \emptyset \to \mathcal{K}$ is a randomized algorithm that takes no input and returns a fresh secret key K.
2. $\mathsf{Enc} : (\mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}) \to (\mathcal{C} \cup \{\perp\})$ is a deterministic algorithm that takes a 4-tuple of a key K, nonce N, associated data A, and message M and returns a ciphertext C or an error (denoted by $\perp$).

3. $\mathsf{Dec} : (\mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{C}) \rightarrow (\mathcal{M} \cup \{\perp\})$ is a deterministic algorithm that takes a 4-tuple of a key K, nonce N, associated data A, and ciphertext C and returns a plaintext M or an error (denoted by $\perp$).

We call the non-message inputs to Enc—the key, nonce, and associated data—the *encryption context* and the non-ciphertext inputs to Dec—the key, nonce, and associated data—the *decryption context*. And, for a given message, say that an encryption context is *valid* if Enc succeeds (i.e., does not output $\perp$). Similarly, for a given ciphertext, say that a decryption context is *valid* if Dec succeeds (i.e., does not output $\perp$).

For traditional AEAD correctness, we need Enc to be the inverse of Dec. In other words, for any 4-tuple $(K, N, A, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$, it holds that

$$\mathsf{Dec}(K, N, A, \mathsf{Enc}(K, N, A, M)) = M\,.$$

In addition, we impose *tidyness* [133], *ciphertext validity*, and *length uniformity* assumptions. Tidyness requires that for any 4-tuple $(K, N, A, C) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{C}$, it holds that

$$\mathsf{Dec}(K, N, A, C) = M \neq \perp \implies \mathsf{Enc}(K, N, A, M) = C\,.$$

Ciphertext validity requires that for every ciphertext $C \in \mathcal{C}$ there exists at least one valid decryption context $(K, N, A) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A}$; that is $\mathsf{Dec}(K, N, A, C) \neq \perp$. Length uniformity requires that the length of a ciphertext depends only on the length of the message and the length of the associated data.

Finally, for AEAD security, we use the traditional *privacy* and *authenticity* definitions [145, §3].

Committing authenticated encryption. A number of prior notions for committing AEAD have been proposed. In Figure 2.3 we provide the CMT-1 and CMT-3 games from

```
CMT-1($\mathcal{A}$):
$((K_1, N_1, A_1), (K_2, N_2, A_2), C) \leftarrow\!\!\$\ \mathcal{A}$
$M_1 \leftarrow \mathsf{AEAD.Dec}(K_1, N_1, A_1, C)$
$M_2 \leftarrow \mathsf{AEAD.Dec}(K_2, N_2, A_2, C)$
// decryption success
If $M_1 = \bot$ or $M_2 = \bot$
    Return false
// commitment condition
If $K_1 = K_2$
    Return false
Return true
```

```
CMT-3($\mathcal{A}$):
$((K_1, N_1, A_1), (K_2, N_2, A_2), C) \leftarrow\!\!\$\ \mathcal{A}$
$M_1 \leftarrow \mathsf{AEAD.Dec}(K_1, N_1, A_1, C)$
$M_2 \leftarrow \mathsf{AEAD.Dec}(K_2, N_2, A_2, C)$
// decryption success
If $M_1 = \bot$ or $M_2 = \bot$
    Return false
// commitment condition
If $(K_1, N_1, A_1) = (K_2, N_2, A_2)$
    Return false
Return true
```

Figure 2.3: The CMT-1 and CMT-3 games from [16], with differences between them are highlighted.

Bellare and Hoang [16]. The FROB game from Farshim, Orlandi, and Rosie [78] adapted to the AEAD setting by Grubbs, Lu, and Ristenpart [87], is the same except that the final highlighted predicate is changed to "$K_1 = K_2$ or $N_1 \neq N_2$". The FROB game asks the adversary to produce a ciphertext that decrypts under two different keys with the same nonce. The CMT-1 game is more permissive and removes the condition that the nonce be the same. The CMT-3 game is even more permissive and relaxes the different key condition to different keys, nonces, or associated data. Bellare and Hoang [16] show that CMT-3 implies CMT-1, which implies FROB. We will expand on these definitions with a more general framework next.

2.2 Granular Committing Encryption Definitions

We provide a more general framework for defining commitment security for encryption. As motivation, we observe that while the CMT-1 and the stronger CMT-3 notions provide good security goals for constructions, they do not precisely capture the *way* in which attacks violate security—which parts of the decryption context does the attacker need to control, which parts have been pre-selected by some other party, and which parts are known to the attacker.

These considerations are crucial for determining the exploitability of commitment vulnerabilities in practice. For instance, the vulnerability in Facebook attachment franking [77] exploited by Dodis et al. [71, §3] only works if the nonces are the same; and the key rotation attack described by Albertini et al. [3] only works with keys previously imported to the key management service. And, looking ahead, we propose a variant of the Subscribe with Google attack described by Albertini et al. [3] in which a malicious publisher provides a full decryption context only knowing the honestly published ciphertext.

We provide a more general framework for commitment security notions that more precisely captures attack settings. As we will see in subsequent sections, our definitions provide a clearer explanatory framework for vulnerabilities.

Committing security framework. We find it useful to expand the set of security notions to more granularly capture the ways in which the two decryption contexts are selected that generalizes context commitment security. In Figure 2.4 we detail the CMT[Σ] game, parameterized by a *setting* $\Sigma = (\mathsf{ts}, \mathsf{S}, \mathsf{P})$ that specifies a *target specifier* ts, a *context selector* S, and a *predicate* P (to be defined next.) The adversary helps compute a ciphertext and two decryption contexts $(C, (K_1, N_1, A_1), (K_2, N_2, A_2))$, what we call a *commitment attack instance (cat)*. The adversary wins if C decrypts under both decryption contexts, and the two decryption contexts satisfy the predicate P. The parameterization allows attack settings in terms of which portions of the commitment attack instance are attacker controlled versus chosen in some other way, and which of the latter are revealed to the attacker.

We now provide more details. A commitment attack instance is a tuple $(C, (K_1, N_1, A_1), (K_2, N_2, A_2))$ consisting of a ciphertext $C \in \mathcal{C}$; two keys $K_1, K_2 \in \mathcal{K}$; two nonces $N_1, N_2 \in \mathcal{N}$; and two associated data $A_1, A_2 \in \mathcal{A}$. A target specifier ts

```
CMT[ts, S, P]($\mathcal{A}$):
$\mathrm{cat}_c \leftarrow\!\!\$\ \mathsf{S}$
$\mathrm{cat}_a \leftarrow\!\!\$\ \mathcal{A}(\mathrm{Reveal}_{\mathsf{ts}}(\mathrm{cat}_c))$
$\mathrm{cat} \leftarrow \mathrm{Merge}_{\mathsf{ts}}(\mathrm{cat}_c, \mathrm{cat}_a)$
If $\mathrm{cat} = \perp$:
    Return false
$(C, (K_1, N_1, A_1), (K_2, N_2, A_2)) \leftarrow \mathrm{cat}$
$M_1 \leftarrow \mathsf{AEAD.Dec}(K_1, N_1, A_1, C)$
$M_2 \leftarrow \mathsf{AEAD.Dec}(K_2, N_2, A_2, C)$
If $M_1 = \perp$ or $M_2 = \perp$:
    Return false
Return $\mathsf{P}((K_1, N_1, A_1), (K_2, N_2, A_2))$
```

Notion	Predicate P
$\mathrm{CMT}_{\mathsf{k}}$	$(K_1 \neq K_2)$
$\mathrm{CMT}_{\mathsf{n}}$	$(N_1 \neq N_2)$
$\mathrm{CMT}_{\mathsf{a}}$	$(A_1 \neq A_2)$
$\mathrm{CMT}^*_{\mathsf{k}}$	$(K_1 \neq K_2) \wedge (N_1, A_1) = (N_2, A_2)$
$\mathrm{CMT}^*_{\mathsf{n}}$	$(N_1 \neq N_2) \wedge (K_1, A_1) = (K_2, A_2)$
$\mathrm{CMT}^*_{\mathsf{a}}$	$(A_1 \neq A_2) \wedge (K_1, N_1) = (K_2, N_2)$

Figure 2.4: **(Left)** The CMT[Σ] commitment security game, parameterized by $\Sigma = (\mathsf{ts}, \mathsf{S}, \mathsf{P})$, a target selector ts, context selector S, and predicate P. **(Right)** Predicates for the permissive notions $\mathrm{CMT}_{\mathsf{k}}$, $\mathrm{CMT}_{\mathsf{n}}$, $\mathrm{CMT}_{\mathsf{a}}$ and restrictive notions $\mathrm{CMT}^*_{\mathsf{k}}$, $\mathrm{CMT}^*_{\mathsf{n}}$, $\mathrm{CMT}^*_{\mathsf{a}}$, where $\mathsf{ts} = \emptyset$.

is a subset of labels $\{\mathsf{C}, \mathsf{k}_1, \mathsf{n}_1, \mathsf{a}_1, \mathsf{k}_2, \mathsf{n}_2, \mathsf{a}_2\} \times \{\cdot, \hat{\cdot}\}$. The left set labels the components of a commitment attack instance, called component labels, and the right set denotes whether the specified component is revealed to the adversary (no hat means revealed and hat means not revealed.) For example, $\mathsf{ts} = \{\mathsf{k}_1, \hat{\mathsf{k}}_2\}$ indicates the K_1 and K_2 in the context, and that K_1 is revealed to the attacker.

A context selector S is a randomized algorithm that takes no input and produces the challenger-defined elements of a commitment attack instance, denoted cat_c, as specified by the target specifier ts. The reveal function $\mathrm{Reveal}_{\mathsf{ts}}$ parameterized by ts, takes a subset of a commitment attack instance and reveals the components that ts tells it to reveal; i.e., the specified components with no hat. The merge function $\mathrm{Merge}_{\mathsf{ts}}(\mathrm{cat}_c, \mathrm{cat}_a)$ parameterized by the target specifier ts, takes two subsets of commitment attack instances cat_c (challenger-defined elements) and cat_a (adversary-defined elements) and works as follows. First, it checks for every component specified by ts that cat_c has a corresponding value. Second, it checks that for every component specified by ts, if cat_a has a value, that it matches the value in cat_c. If either of these checks fail, it outputs

$\perp$. Otherwise, it returns their union $\mathrm{cat}_c \cup \mathrm{cat}_a$. Finally, the predicate P takes two decryption contexts output by $\mathrm{Merge}_{\mathsf{ts}}(\mathrm{cat}_c, \mathrm{cat}_a)$, and outputs true if they satisfy some criteria (e.g., that $K_1 \neq K_2$), and false otherwise.

We associate to a setting $\Sigma = (\mathsf{ts}, \mathsf{S}, \mathsf{P})$, AEAD Π, and adversary $\mathcal{A}$ the CMT advantage defined as

$$\mathbf{Adv}_{\Pi}^{\mathrm{CMT}[\Sigma]}(\mathcal{A}) := \Pr[\,\mathrm{CMT}[\Sigma](\mathcal{A}) \Rightarrow \mathsf{true}\,]\,.$$

Taking a concrete security approach, we will track the running time used by $\mathcal{A}$ and provide explicit advantage functions. Adapting our notions to support asymptotic definitions of security is straightforward: in our discussions we will often say a scheme is CMT[Σ] secure as informal shorthand that no adversary can win the CMT[Σ] game with "good" probability using "reasonable" running time.

Capturing CMT-1, CMT-3, and more via predicates To understand our definitional framework further, we can start by seeing how to instantiate it to coincide with prior notions. Let $\mathsf{ts} = \emptyset$ indicate the empty target selector, meaning that $\mathcal{A}$ chooses the ciphertext and two decryption contexts fully. Then the set of Σ settings that use the empty target selector defines a family of security goals, indexed solely by predicates, which we denote by CMT[P]. This family includes CMT-1 by setting $\mathsf{P} := (K_1 \neq K_2)$ and CMT-3 by setting $\mathsf{P} := (K_1, N_1, A_1) \neq (K_2, N_2, A_2)$. Not all instances in this family are interesting: consider, for example, when P always outputs true or false. Nevertheless, the flexibility here allows for more granular specification of adversarial ability. For instance, the predicate that requires $(K_1 \neq K_2) \wedge (N_1 = N_2)$ captures a setting like that of the Dodis et al. [71] attack against Facebook's message franking, which requires that both decryption contexts have the same nonce.

Three games of particular interest are those with predicates that focus on inequality of the three individual context components: $(K_1 \neq K_2)$, $(N_1 \neq N_2)$, and $(A_1 \neq A_2)$. For notational brevity, we let game $\text{CMT}_{\mathsf{k}} := \text{CMT}[\mathsf{P} = (K_1 \neq K_2)]$ and similarly $\text{CMT}_{\mathsf{n}} := \text{CMT}[\mathsf{P} = (N_1 \neq N_2)]$ and $\text{CMT}_{\mathsf{a}} := \text{CMT}[\mathsf{P} = (A_1 \neq A_2)]$. Then CMT_{k} corresponds to CMT-1, but CMT_{n} and CMT_{a} are new. They are also orthogonal to CMT-1, in the sense that we can give schemes that achieve CMT-1 but not CMT_{a} nor CMT_{n} security (see Theorem 9 in the full version [126].) All three are, however, implied by being CMT-3 secure, and a scheme that simultaneously meets CMT_{k}, CMT_{n}, and CMT_{a} also enjoys CMT-3 security (see Lemmas 7 and 8 in the full version [126].)

Note that CMT_{k}, CMT_{n}, and CMT_{a} are *permissive*: as long as the relevant component is distinct across the two contexts, it does not matter whether the other components are distinct. Also, of interest are *restrictive* versions; for example, we can consider $\text{CMT}^*_{\mathsf{k}} := \text{CMT}[(K_1 \neq K_2) \wedge (N_1, A_1) = (N_2, A_2)]$ which requires that the nonces and associated data are the same. Similarly, we can define restrictive notions $\text{CMT}^*_{\mathsf{n}}$ and $\text{CMT}^*_{\mathsf{a}}$. Restrictive versions are useful as they correspond to attacks that have limited control over the decryption context. Interestingly, these restrictive notions are not equivalent to the corresponding permissive notions, nor does a scheme that simultaneously meets $\text{CMT}^*_{\mathsf{k}}$, $\text{CMT}^*_{\mathsf{n}}$, and $\text{CMT}^*_{\mathsf{a}}$ achieve CMT-3 security (see Theorem 10 in the full version [126].)

Targeted attacks. Returning to settings with target specifier $\mathsf{ts} \neq \emptyset$, we can further increase the family of notions considered to capture situations where a portion of the context is pre-selected. For instance, in the key rotation example of Albertini et al. [3] mentioned earlier, we would have $\mathsf{ts} = \{\mathsf{k}_1, \mathsf{k}_2\}$ and $\mathsf{S} = \{K_1 \leftarrow\$ \mathcal{K}; K_2 \leftarrow\$ \mathcal{K}; \text{Return } (K_1, K_2)\}$ to indicate that the malicious sender has to use the two randomly generated keys.

However, not all targeted attack settings are interesting. For some target specifiers

ts, we can specify a context selector S such that no adversary can achieve non-zero advantage. In particular, if we have $\mathsf{ts} = \{\mathsf{C}, \mathsf{k}_1, \mathsf{n}_1, \mathsf{a}_1\}$ and have S pick ciphertext C and context (K_1, N_1, A_1) such that $\mathsf{AEAD.Dec}(K_1, N_1, A_1, C)$ returns $\perp$, then no adversary can win the game, making the security notion trivial (all schemes achieve it.)

Hiding target components. Finally, our game considers target specifiers ts that indicate that some values chosen by S should remain hidden from $\mathcal{A}$. For example, the Subscribe with Google attack described by Albertini et al. [3] can be reframed as a meddler-in-the-middle attack as follows. A publisher creates premium content M_1 and encrypts it using a context (K_1, N_1, A_1) to get a ciphertext C. The ciphertext C is published, but the context (K_1, N_1, A_1) is hidden. A malicious third-party, only looking at the ciphertext C, tries to construct a valid decryption context (K_2, N_2, A_2) and uses that to sell fake paywall bypasses. We can formalize this setting by having the target specifier $\mathsf{ts} = \{\mathsf{C}, \hat{\mathsf{k}}_1, \hat{\mathsf{n}}_1, \hat{\mathsf{a}}_1\}$, with the context selector S as

$$K_1 \leftarrow\!\!{\scriptstyle\$}\ \mathcal{K};\ \ N_1 \leftarrow\!\!{\scriptstyle\$}\ \mathcal{N};\ \ A_1 \leftarrow\!\!{\scriptstyle\$}\ \mathcal{A};\ \ M_1 \leftarrow\!\!{\scriptstyle\$}\ \mathcal{M}$$
$$\text{Return } (\mathsf{AEAD.Enc}(K_1, N_1, A_1, M_1), K_1, N_1, A_1)$$

and with $\mathrm{Reveal}_{\mathsf{ts}}(C, K_1, N_1, A_1)$ outputting C.

Context discoverability security. Dodis et al [71, §5] and Albertini et al. [3, §3.3] have pointed out that traditional CMT games are analogous to collision-resistance for hash functions, in the sense that the goal is to find two different *encryption contexts* (K_1, N_1, A_1, M_1) and (K_2, N_2, A_2, M_2) such that they produce the same ciphertext C. Under this lens, CMT with targeting (and no hiding) is like second preimage resistance, and CMT with targeting and hiding is like preimage resistance. But, the analogy to preimage resistance is not perfect, since we are not asking for *any* preimage but rather one that is not the same as the original. Further, this restriction is unnecessary. Going back to the meddler-in-the-middle example above, it suffices for an on-path attacker to

```
CDY[ts, S](A):
    dat_c ←$ S
    dat_a ←$ A(Reveal_ts(t))
    dat ← Merge_ts(dat_c, dat_a)
    If dat = ⊥:
        Return false
    (C, (K, N, A)) ← dat
    M ← AEAD.Dec(K, N, A, C)
    If M = ⊥:
        Return false
    Return true
```

```
CDY[{k, n}, S](A):
    (C, K, N) ←$ S
    A ←$ A(C, K, N)
    M ← AEAD.Dec(K, N, A, C)
    If M = ⊥:
        Return false
    Return true
```

```
CDY[{k, a}, S](A):
    (C, K, A) ←$ S
    N ←$ A(C, K, A)
    M ← AEAD.Dec(K, N, A, C)
    If M = ⊥:
        Return false
    Return true
```

Figure 2.5: **(Left)** The CDY[ts, S] commitment security game, parameterized by a target specifier ts and a context selector S. **(Middle)** The variant of CDY[Σ] used in the definition of CDY^*_{a}. **(Right)** The variant of CDY[Σ] used in the definition of CDY^*_{n}.

produce any valid context. Thus, we find it useful to define a new preimage resistance-inspired notion of commitment security.

In Figure 2.5 we define the game CDY[ts, S], parameterized by a setting $\Sigma = (\text{ts}, \text{S})$ that specifies a target specifier ts and a context selector S. In more detail, a *discoverability attack instance (dat)* is a ciphertext and a decryption context $(C, (K, N, A))$. Here, a target specifier ts is a subset of $\{\text{k}, \text{n}, \text{a}\} \times \{\cdot, \hat{\cdot}\}$ and a context selector S is a randomized algorithm that takes no input and produces a ciphertext and the elements of a decryption context specified by the target specifier ts. The reveal function $\text{Reveal}_{\text{ts}}$ and the merge function $\text{Merge}_{\text{ts}}(\text{dat}_c, \text{dat}_a)$ work similarly to their CMT counterparts. Finally, the goal of the adversary is to produce *one* valid decryption context for the target ciphertext.

We associate to a setting $\Sigma = (\text{ts}, \text{S})$, AEAD scheme Π, and adversary $\mathcal{A}$ the CDY advantage defined as

$$\mathbf{Adv}_{\Pi}^{\text{CDY}[\Sigma]}(\mathcal{A}) = \Pr[\, \text{CDY}[\Sigma](\mathcal{A}) \Rightarrow \text{true} \,] \,.$$

Restricted CDY and its variants. To more accurately capture attack settings and to prove relations, we find it useful to define restricted variants of the CDY[Σ] game. A class of games of particular interest are ones that allow targeting under *any* context selector; we call this class *restricted CDY*. For a target specifier ts, let $\mathrm{CDY}^*[\mathsf{ts}]$ be the game where the adversary is given a ciphertext and elements of a decryption context specified by ts, all selected arbitrarily, and needs to produce the remaining elements of a decryption context such that $\mathsf{AEAD.Dec}(K, N, A, C) \neq \bot$. Formally, for an AEAD scheme Π and adversary $\mathcal{A}$, we define the CDY^* advantage as

$$\mathbf{Adv}_{\Pi}^{\mathrm{CDY}^*[\mathsf{ts}]}(\mathcal{A}) = \Pr\left[\text{ for all } \mathsf{S}, \mathrm{CDY}[\mathsf{ts}, \mathsf{S}](\mathcal{A}) \Rightarrow \mathsf{true}\right].$$

In addition, we find it useful to define three specific variants of CDY^* that allow targeting two-of-three components of a decryption context. Let $\mathrm{CDY}^*_{\mathsf{a}}$ be the game where the adversary is given an arbitrary ciphertext C, key K, and nonce N, and has to produce associated data A such that $\mathsf{AEAD.Dec}(K, N, A, C) \neq \bot$. Formally, for an AEAD scheme Π and adversary $\mathcal{A}$, we define the $\mathrm{CDY}^*_{\mathsf{a}}$ advantage as

$$\mathbf{Adv}_{\Pi}^{\mathrm{CDY}^*_{\mathsf{a}}}(\mathcal{A}) = \Pr\left[\text{ for all } \mathsf{S}, \mathrm{CDY}[\{\mathsf{k}, \mathsf{n}\}, \mathsf{S}](\mathcal{A}) \Rightarrow \mathsf{true}\right].$$

The $\mathrm{CDY}^*_{\mathsf{k}}$ and $\mathrm{CDY}^*_{\mathsf{n}}$ games are defined similarly where the adversary has to produce a valid key and nonce respectively such that decryption succeeds when the remaining inputs to decryption are pre-selected. Formally, for an AEAD scheme Π and adversary $\mathcal{A}$, we define the $\mathrm{CDY}^*_{\mathsf{k}}$ and $\mathrm{CDY}^*_{\mathsf{n}}$ advantage as

$$\mathbf{Adv}_{\Pi}^{\mathrm{CDY}^*_{\mathsf{k}}}(\mathcal{A}) = \Pr\left[\text{ for all } \mathsf{S}, \mathrm{CDY}[\{\mathsf{n}, \mathsf{a}\}, \mathsf{S}](\mathcal{A}) \Rightarrow \mathsf{true}\right],$$
$$\mathbf{Adv}_{\Pi}^{\mathrm{CDY}^*_{\mathsf{n}}}(\mathcal{A}) = \Pr\left[\text{ for all } \mathsf{S}, \mathrm{CDY}[\{\mathsf{k}, \mathsf{a}\}, \mathsf{S}](\mathcal{A}) \Rightarrow \mathsf{true}\right].$$

Note that the context selector can only select *valid* ciphertexts, which sidesteps issues with formatting. Without this constraint, a context selector could select a ciphertext that has invalid padding for a scheme that requires valid padding, thereby making the notion trivial (all schemes achieve it.)

Furthermore, specific variants like $\text{CDY}^*_{\mathsf{a}}$ may be trivial even with this constraint. For instance, if the ciphertext embeds the nonce, then one can pick some key K, some ciphertext C embedding some nonce N_1, some other nonce N_2, then no $\text{CDY}^*_{\mathsf{a}}$ adversary can pick associated data A such that C decrypts correctly under (K, N_2, A). However, in the context of this restricted CDY notion, we think this is desired behavior and delegate capturing nuances like this to the unrestricted CDY notion (which can capture this by restricting to context selectors which ensure that the nonce embedded is the same as the nonce provided.)

With context compression, CMT-3 implies restricted CDY A CDY[Σ] attack does not always imply a CMT[Σ] attack. Consider, for example, the "identity" AEAD that has $\mathsf{Enc}(K, N, A, M) \Rightarrow K \parallel N \parallel A \parallel M$ which has an immediate CDY[Σ] attack but is CMT[Σ] secure since a ciphertext can only be decrypted under one context.[2] However, continuing with the hash function analogy, we wonder if a "compression" assumption could make this implication hold. In Theorem 1 we show this statement for $\text{CDY}^*[\mathsf{ts} = \emptyset]$ and CMT-3. And note that this generalizes to $\text{CDY}^*[\mathsf{ts}]$ for any ts with an appropriate compression assumption. Notably, it holds for $\text{CDY}^*_{\mathsf{a}}$ if we assume compression over associated data rather than the full context.

Theorem 1. *Fix some AEAD* Π*. Then for any adversary* $\mathcal{A}$ *that wins the* $\text{CDY}^*[\mathsf{ts} = \emptyset]$ *game, we can give an adversary* $\mathcal{B}$ *such that*

$$\mathbf{Adv}_{\Pi}^{\text{CDY}^*[\mathsf{ts}=\emptyset]}(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}_{\Pi}^{\text{CMT-3}}(\mathcal{B}) + \text{ProbBadCtx}_{\Pi}\,, \tag{2.2}$$

where ProbBadCtx_{Π} *is the probability that a random decryption context, when used for*

[2] While the "identity" AEAD is not secure in the sense of privacy [145, §3], one can construct a secure counterexample by using a wide pseudorandom permutation [33].

```
$\mathcal{B}$:
$K_1 \leftarrow\!\$\, \mathcal{K}$; $N_1 \leftarrow\!\$\, \mathcal{N}$; $A_1 \leftarrow\!\$\, \mathcal{A}$
$M_1 \leftarrow\!\$\, \mathcal{M}$
$\mathrm{ctx}_1 \leftarrow (K_1, N_1, A_1)$
$C \leftarrow \Pi.\mathsf{Enc}(K_1, N_1, A_1, M_1)$
$\mathrm{ctx}_2 \leftarrow\!\$\, \mathcal{A}(C)$
If $\mathrm{ctx}_2 = \bot$:
    Return $\bot$
$(K_1, N_2, A_2) \leftarrow \mathrm{ctx}_2$
If $(K_1, N_1, A_1) = (K_2, N_2, A_2)$
    Return $\bot$
Return
$(C, (K_1, N_1, A_1), (K_2, N_2, A_2))$
```

```
$\mathsf{S}$:
$K_1 \leftarrow\!\$\, \mathcal{K}$; $N_1 \leftarrow\!\$\, \mathcal{N}$; $A_1 \leftarrow\!\$\, \mathcal{A}$
$M_1 \leftarrow\!\$\, \mathcal{M}$
$C \leftarrow \Pi.\mathsf{Enc}(K_1, N_1, A_1, M_1)$
Return $C$
```

Figure 2.6: Pseudocode for the CMT-3 adversary $\mathcal{B}$ and CDY* context selector S, used in proof of Theorem 1.

encrypting a random message, is the only valid decryption context for the resulting ciphertext.

Proof. This proof is adapted from Bellare and Rogaway [25, p.147], where they prove a similar theorem for hash functions. We construct an adversary $\mathcal{B}$ that randomly samples a context (K_1, N_1, A_1), encrypts a random message to get a ciphertext C, then asks the CDY adversary $\mathcal{A}$ to produce a decryption context for C to get (K_2, N_2, A_2). This ciphertext generation can be viewed as a valid CDY context selector S so $\mathcal{B}$ wins if the returned context is different from the one it sampled; i.e., $(K_1, N_1, A_1) \neq (K_2, N_2, A_2)$. The pseudocode for $\mathcal{B}$ and S is given in Figure 2.6 and the success probability is analyzed below.

Per the above discussion the advantage of $\mathcal{B}$ is

$$\mathbf{Adv}_{\Pi}^{\text{CMT-3}}(\mathcal{B}) = \Pr[(\mathcal{A}(C) \neq \bot) \wedge (\mathrm{ctx}_1 \neq \mathrm{ctx}_2)]\,, \tag{2.3}$$

where without loss of generality, we are assuming that $\mathcal{A}$ always produces a valid context or fails and produces $\bot$. But, before simplifying this equation, we need to define some terminology. First, let us define the set of valid decryption contexts for a ciphertext as

$$\Gamma(C) := \{(K, N, A) : (\Pi.\mathsf{Dec}(K, N, A, C) \neq \bot)\}.$$

Now, for a given message M, let us also define the set of "bad" decryption contexts which when used for encrypting M, remain the only valid decryption context for the resulting ciphertext

$$\mathrm{BadCtxs}(M) := \{(K, N, A) : |\Gamma(\Pi.\mathsf{Enc}(K, N, A, M))| = 1\}.$$

Finally, let us define the probability that a random decryption context is bad

$$\mathrm{ProbBadCtx}_{\Pi} := \Pr[(K, N, A) \in \mathrm{BadCtxs}(M)],$$

over the choice $(K, N, A, M) \leftarrow\!\!\$\ (\mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M})$. Using this notation we can rewrite Equation 2.3, where the probabilities are over the choice $(K, N, A, M) \leftarrow\!\!\$\ (\mathcal{K}\times\mathcal{N}\times\mathcal{A}\times\mathcal{M})$, as

$$\begin{aligned}\mathbf{Adv}^{\text{CMT-3}}_{\Pi}(\mathcal{B}) &= \Pr[(\mathcal{A}(C) \neq \bot) \wedge (\mathrm{ctx}_1 \neq \mathrm{ctx}_2)] \\ &\geq \Pr[(\mathcal{A}(C) \neq \bot) \wedge (\mathrm{ctx}_1 \neq \mathrm{ctx}_2) \wedge (\mathrm{ctx}_1 \notin \mathrm{BadCtxs}(M))].\end{aligned}$$

Using conditional probability, we can rewrite this term as

$$\Pr[\mathrm{ctx}_1 \neq \mathrm{ctx}_2 \mid (\mathcal{A}(C) \neq \bot) \wedge (\mathrm{ctx}_1 \notin \mathrm{BadCtxs}(m))] \cdot \Pr[(\mathcal{A}(C) \neq \bot) \wedge (\mathrm{ctx}_1 \notin \mathrm{BadCtxs}(m))].$$

Recall that if $\mathrm{ctx}_1 \notin \mathrm{BadCtxs}(m)$, then the adversary must choose one of at least two valid contexts, each of which are equally likely to be ctx_1 (even conditioned on C). Thus the probably that it picks ctx_1 is at most $1/2$, and so

$$\begin{aligned}\mathbf{Adv}^{\text{CMT-3}}_{\Pi}(\mathcal{B}) &\geq \frac{1}{2} \cdot \Pr[(\mathcal{A}(C) \neq \bot) \wedge (\mathrm{ctx}_1 \notin \mathrm{BadCtxs}(m))] \\ &\geq \frac{1}{2} \cdot (\Pr[\mathcal{A}(C) \neq \bot] - \Pr[\mathrm{ctx}_1 \in \mathrm{BadCtxs}(m)]).\end{aligned}$$

```
$\mathcal{B}$:
$K_1 \leftarrow\$ \mathcal{K}$; $N_1 \leftarrow\$ \mathcal{N}$; $A_1 \leftarrow\$ \mathcal{A}$
$M_1 \leftarrow\$ \mathcal{M}$
$C \leftarrow \Pi.\mathsf{Enc}(K_1, N_1, A_1, M_1)$
$K_2 \leftarrow K_1 + 1$; $N_2 \leftarrow N_1 + 1$
$A_2 \leftarrow\$ \mathcal{A}(C, K_2, N_2)$
If $A_2 = \bot$
    Return $\bot$
Return $(C, (K_1, N_1, A_1), (K_2, N_2, A_2))$
```

```
$\mathsf{S}$:
$K_1 \leftarrow\$ \mathcal{K}$; $N_1 \leftarrow\$ \mathcal{N}$; $A_1 \leftarrow\$ \mathcal{A}$
$M_1 \leftarrow\$ \mathcal{M}$
$C \leftarrow \Pi.\mathsf{Enc}(K_1, N_1, A_1, M_1)$
$K_2 \leftarrow K_1 + 1$; $N_2 \leftarrow N_1 + 1$
Return $(C, K_2, N_2)$
```

Figure 2.7: Pseudocode for the CMT-3 adversary $\mathcal{B}$ and $\text{CDY}^*_{\mathsf{a}}$ context selector S, used in proof of Theorem 2.

Putting it all together, we get that

$$\mathbf{Adv}^{\text{CMT-3}}_{\Pi}(\mathcal{B}) \geq \frac{1}{2} \cdot \left(\mathbf{Adv}^{\text{CDY}^*[\mathsf{ts}=\emptyset]}_{\Pi}(\mathcal{A}) - \text{ProbBadCtx}_{\Pi}\right),$$

and finally rearranging gives the desired result. □

CMT-3 implies restricted variants of CDY We now show that if an attack against any of $\text{CDY}^*_{\mathsf{k}}$, $\text{CDY}^*_{\mathsf{n}}$, or $\text{CDY}^*_{\mathsf{a}}$ implies an attack against CMT-3. Theorem 2 shows this for $\text{CDY}^*_{\mathsf{a}}$, but it readily generalizes to $\text{CDY}^*_{\mathsf{k}}$ and $\text{CDY}^*_{\mathsf{n}}$.

Theorem 2. *Fix some AEAD Π with key space $|\mathcal{K}| \geq 2$ and nonce space $|\mathcal{N}| \geq 2$. Then for any adversary $\mathcal{A}$ that wins the $\text{CDY}^*_{\mathsf{a}}$ game, we can give an adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\text{CDY}^*_{\mathsf{a}}}_{\Pi}(\mathcal{A}) = \mathbf{Adv}^{\text{CMT-3}}_{\Pi}(\mathcal{B}),$$

and the runtime of $\mathcal{B}$ is that of $\mathcal{A}$.

Proof. We prove this by constructing $\mathcal{B}$ such that it succeeds whenever $\mathcal{A}$ succeeds. The adversary $\mathcal{B}$ randomly samples a context (K_1, N_1, A_1), encrypts a random message to get a ciphertext C, selects some other key K_2 and nonce N_2 and asks the $\text{CDY}^*_{\mathsf{a}}$ adversary $\mathcal{A}$ to produce an associated data A_2 such that (K_2, N_2, A_2) can decrypt C. This ciphertext and partial context construction can be viewed as a valid context selector S.

The pseudocode for the adversary $\mathcal{B}$ and the context selector S are given in Figure 2.7. And, notice that by construction, $\mathcal{B}$ wins whenever $\mathcal{A}$ succeeds. $\square$

This approach of constructing $\mathcal{B}$ readily generalizes to $\mathrm{CDY}^*_{\mathsf{n}}$ and $\mathrm{CDY}^*_{\mathsf{k}}$. Further, notice that the $\mathcal{B}$ constructed in Figure 2.7 wins $\mathrm{CMT}_{\mathsf{k}}$ and $\mathrm{CMT}_{\mathsf{n}}$; and similar relations hold for adversaries $\mathcal{B}$ constructed from $\mathrm{CDY}^*_{\mathsf{n}}$ and $\mathrm{CDY}^*_{\mathsf{k}}$ adversaries. Corollary 3 captures these implications.

Corollary 3. *Fix some AEAD Π with key space $|\mathcal{K}| \geq 2$, nonce space $|\mathcal{N}| \geq 2$, and associated data space $|\mathcal{A}| \geq 2$. Then the following three statements hold. First, for any adversary $\mathcal{A}_1$ that wins the $\mathrm{CDY}^*_{\mathsf{a}}$ game, we can give an adversary $\mathcal{B}_1$ such that*

$$\mathbf{Adv}_{\Pi}^{\mathrm{CDY}^*_{\mathsf{a}}}(\mathcal{A}_1) = \mathbf{Adv}_{\Pi}^{\mathrm{CMT}_{\mathsf{k}}}(\mathcal{B}_1) = \mathbf{Adv}_{\Pi}^{\mathrm{CMT}_{\mathsf{n}}}(\mathcal{B}_1)\,.$$

Second, for any adversary $\mathcal{A}_2$ that wins the $\mathrm{CDY}^*_{\mathsf{n}}$ game, we can give an adversary $\mathcal{B}_2$ such that

$$\mathbf{Adv}_{\Pi}^{\mathrm{CDY}^*_{\mathsf{n}}}(\mathcal{A}_2) = \mathbf{Adv}_{\Pi}^{\mathrm{CMT}_{\mathsf{k}}}(\mathcal{B}_2) = \mathbf{Adv}_{\Pi}^{\mathrm{CMT}_{\mathsf{a}}}(\mathcal{B}_2)\,.$$

Third, for any adversary $\mathcal{A}_3$ that wins the $\mathrm{CDY}^*_{\mathsf{k}}$ game, we can give an adversary $\mathcal{B}_3$ such that

$$\mathbf{Adv}_{\Pi}^{\mathrm{CDY}^*_{\mathsf{k}}}(\mathcal{A}_3) = \mathbf{Adv}_{\Pi}^{\mathrm{CMT}_{\mathsf{n}}}(\mathcal{B}_3) = \mathbf{Adv}_{\Pi}^{\mathrm{CMT}_{\mathsf{a}}}(\mathcal{B}_3)\,.$$

And the runtimes of $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$ are that of $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$, respectively.

2.3 Context Discovery Attacks against AEAD

We show context discovery attacks on many AEAD schemes which delegate their authenticity to a non-preimage resistant MAC. Specifically, we show $\mathrm{CDY}^*_{\mathsf{a}}$ attacks on

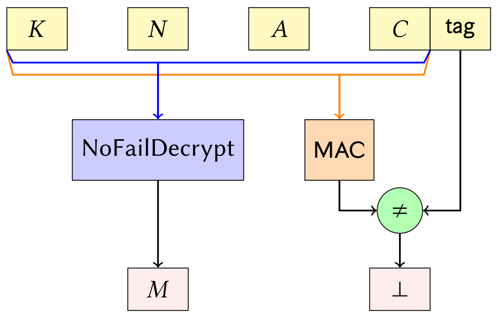

Figure 2.8: Decryption structure of AEAD schemes which delegate their authenticity to a MAC. Should the MAC tag comparison fail, the routine outputs an error ($\perp$), otherwise a message is always output by NoFailDecrypt.

EAX [28], SIV [150], CCM [74], GCM [73], and OCB3 [113], and CDY$_n^*$ attacks on EAX [28] and GCM [73].

We say that an AEAD delegates its authenticity to a MAC if during decryption, a message is output whenever the MAC comparison succeeds. To formalize this, we define NoFailDecrypt as a class of decryption algorithms that never fail. In other words, given a key, nonce, associated data, and ciphertext, they always produce a message. For example, ECB and CTR decryption are NoFailDecrypt algorithms since a valid ciphertext decrypts under any choice of key, nonce, and associated data. On the other hand, CBC with PKCS7 padding is not a NoFailDecrypt algorithm since there most ciphertexts do not decrypt under all choices of key, nonce, and associated data because the decrypted plaintext has incorrect padding.

With this terminology, we say that an AEAD delegates its authenticity to a MAC if it can be written as a combination of a MAC and a NoFailDecrypt algorithm such that if the MAC check fails, decryption fails; if instead the MAC check passes, then decryption outputs the result of NoFailDecrypt (which never fails). This structure is illustrated in Figure 2.8. As a concrete example, for EAX [28] (described in Figure 2.9), the MAC corresponds to checking the OMAC tag, and the NoFailDecrypt corresponds to the

CTR decryption. In this section, we are particularly interested in schemes that compose this structure with a non-preimage resistant MAC like CMAC [75], GMAC [73, §6.4], or OMAC [28, Fig 1].

The $\text{CDY}_{\mathsf{a}}^{*}$ attacks we show on these schemes have the following outline. Following the definition of the game, the challenger provides the adversary with a ciphertext $C \| \mathsf{tag}$, a target key K, and a target nonce N, and asks it to find an associated data A such that $\text{Decrypt}(K, N, A, C \| \mathsf{tag}) \neq \perp$. Then, the adversary exploits the lack of preimage resistance to find an associated data A such that $\mathsf{MAC}(K, N, A, C) = \mathsf{tag}$ and returns A. Since, in these schemes, the tag check passing guarantees decryption success, we get that decryption succeeds.

For EAX [28] and GCM [73], we also show $\text{CDY}_{\mathsf{n}}^{*}$ attacks. They proceed in a similar fashion to the $\text{CDY}_{\mathsf{a}}^{*}$ attacks but now the adversary finds a nonce N such that $\mathsf{MAC}(K, N, A, C) = \mathsf{tag}$. But, when the nonce length is shorter than a block (which is always true with GCM, and may be true with EAX), the $\text{CDY}_{\mathsf{n}}^{*}$ attacks are slower than the $\text{CDY}_{\mathsf{a}}^{*}$ attacks.

The remainder of the section describes the attacks on EAX. The attacks on SIV, CCM, GCM, and OCB3 are in Appendix B of the full version [126].

$\text{CDY}_{\mathsf{a}}^{*}$ and $\text{CDY}_{\mathsf{n}}^{*}$ attacks on EAX We consider EAX over a 128-bit block cipher as defined in Bellare, Rogaway, and Wagner [28]. For simplicity, we restrict to 128-bit tag, 128-bit nonce,[3] and block-aligned messages and associated data. We note however that this is only to make the exposition simpler and is not necessary for the attack. Pseudocode for the scheme with these parameter choices is given in Figure 2.9.

[3]EAX [28, Figure 4] supports an arbitrary length nonce; 128 bits is the default in the popular Tink library [4], see [5].

```
OMAC(K, M):
    // Compute Constants
    $L \leftarrow E_K(0^{128})$
    $B \leftarrow 2 \cdot L$
    // split into $n$-bit blocks
    // & xor $B$ to the last block
    Let $M_1, \ldots, M_m \leftarrow M$
    $M_m \leftarrow M_m \oplus B$
    // CBC-MAC Evaluation
    $C_0 \leftarrow 0^{128}$
    For $i = 1..m$:
        $C_i \leftarrow E_K(C_{i-1} \oplus M_i)$
    Return $C_m$
```

```
EAX-Decrypt(K, N, A, C):
    // Separate the Tag
    $C \,\|\, \mathsf{tag} \leftarrow C$
    // Compute and Check Tag
    $\mathcal{N} \leftarrow \mathsf{OMAC}(K, 0^{128} \,\|\, N)$
    $\mathcal{H} \leftarrow \mathsf{OMAC}(K, 0^{127}1 \,\|\, A)$
    $\mathcal{C} \leftarrow \mathsf{OMAC}(K, 0^{126}10 \,\|\, C)$
    If $\mathsf{tag} \neq (\mathcal{N} \oplus \mathcal{H} \oplus \mathcal{C})$:
        Return $\perp$
    // CTR Decryption
    $r \leftarrow |C|/16$        // num blocks
    For $i = 0..(r-1)$:
        $M_i \leftarrow C_i \oplus E_K(\mathcal{N} + i)$
    Return $M$
```

```
A(C, K, N):
    $C \,\|\, \mathsf{tag} \leftarrow C$
    // Compute $\xi$
    $\xi \leftarrow \mathsf{tag}$
    $\xi \leftarrow \xi \oplus \mathsf{OMAC}_K(0^{128} \,\|\, N)$
    $\xi \leftarrow \xi \oplus \mathsf{OMAC}_K(0^{126}10 \,\|\, C)$
    // Reconstruct $A$ and Return
    $A \leftarrow E_K^{-1}(\xi)$
    $A \leftarrow A \oplus E_K(0^{127}1) \oplus (2 \cdot E_K(0^{128}))$
    Return $(K, N, A)$
```

Figure 2.9: **(Left)** Pseudocode for OMAC [28, Fig 1], used in EAX, with block-aligned inputs. **(Middle)** Pseudocode for EAX Mode [28] decryption with 128-bit tag, 128-bit nonce, and block-aligned messages and associated data. **(Right)** Pseudocode for an $\mathsf{CDY}_{\mathsf{a}}^*$ attack on EAX.

Let's start by contextualizing the $\mathsf{CDY}_{\mathsf{a}}^*$ game. The challenger provides us with an m-block ciphertext $C = C_1 \cdots C_m \| \mathsf{tag}$, a 128-bit target key K, and a 96-bit target nonce N. And the goal is to find a 1-block associated data A such that $\mathsf{EAX\text{-}Decrypt}(K, N, A, C) \neq \perp$. Notice from Figure 2.9 that decryption passing reduces to the tag check passing. In other words, we can rewrite the goal as finding an associated data A such that

$$\mathsf{tag} = \mathsf{OMAC}_K(0^{128} \,\|\, N) \oplus \mathsf{OMAC}_K(0^{126}10 \,\|\, C) \oplus \mathsf{OMAC}_K(0^{127}1 \,\|\, A)\,. \tag{2.4}$$

We can rearrange terms to get

$$\mathsf{OMAC}_K(0^{127}1 \,\|\, A) = \mathsf{tag} \oplus \mathsf{OMAC}_K(0^{128} \,\|\, N) \oplus \mathsf{OMAC}_K(0^{126}10 \,\|\, C)\,.$$

Notice that the right-hand side is composed entirely of known terms, thus we can evaluate it to some constant ξ. Using the assumption that A is 1-block, we can expand OMAC_K to get

$$E_K(E_K(0^{127}1) \oplus A \oplus (2 \cdot E_K(0^{128}))) = \xi\,.$$

Decrypting both sides under K, and solving for A gives

$$A = E_K^{-1}(\xi) \oplus E_K(0^{127}1) \oplus (2 \cdot E_K(0^{128}))\,.$$

The full pseudocode for this attack is given in Figure 2.9.

This attack generalizes to other parameter choices. It works as is against an arbitrary-length message, an arbitrary-length tag, and an arbitrary-length nonce. In addition, this attack can also be adapted as a $\mathrm{CDY}_{\mathsf{n}}^{*}$ attack. We start by rewriting Equation 2.4 as

$$\mathsf{OMAC}_K(0^{128} \parallel N) = \mathsf{tag} \oplus \mathsf{OMAC}_K(0^{126}10 \parallel C) \oplus \mathsf{OMAC}_K(0^{127}1 \parallel C)\,,$$

and solving for N as we did for A above. Since N is 1 block (128 bits), the reduction is similar, and the success probability remains one. If the nonce length was shorter, then assuming an idealized model like the ideal cipher model, the success probability reduces by a multiplicative factor of $2^{-f \cdot 128}$ where f is the fraction of bytes we do not have control over. For example, if we only had control over 14 of the 16 bytes in an encoded block, then the success probability would reduce by 2^{-16}.

This attack can also be adapted to provide partial control over the output plaintext. Notice that the output plaintext is a CTR decryption under the chosen key with the OMAC of the nonce as IV. Assuming an idealized model where the block cipher is an ideal cipher and OMAC is a random function, for every new choice of key and nonce, we get a random output plaintext. So, by trying 2^m key and nonce pairs, we can expect to control m bits of the output plaintext.

```
SIV-1b-Decrypt$(K, C)$:
c $\leftarrow 1^{n-64}01^{31}01^{31}$
$C_1 \parallel \mathsf{tag} \leftarrow C$
$I \leftarrow \mathsf{tag}$
$K_1 \parallel K_2 \leftarrow K$
// CTR Decryption
ctr $\leftarrow I$ & c
$M \leftarrow C_1 \oplus E_{K_2}(\mathsf{ctr})$
// IV Check
$I' \leftarrow \mathsf{CMAC}^*(K_1, M)$
If $I \neq I'$:
    Return $\perp$
Return $M$
```

```
CMAC*$(K, M)$:
$S \leftarrow \mathsf{CMAC}(K, 0^n)$
Return $\mathsf{CMAC}(K, S \oplus M)$

CMAC$(K, X)$:
$K_s \leftarrow 2 \cdot E_K(0^n)$
Return $E_K(K_s \oplus X)$
```

Figure 2.10: **(Left)** Pseudocode for SIV Mode [150] decryption with an n-bit message and no associated data. **(Right)** Pseudocode for CMAC* [150] and CMAC [75] with an n-bit input.

2.4 Restrictive Commitment Attacks via k-Sum Problems

The previous section's $\mathsf{CDY}_\mathsf{a}^*$ and $\mathsf{CDY}_\mathsf{n}^*$ attacks against GCM, EAX, OCB3, SIV, and CCM immediately give rise to *permissive* CMT_k attacks against each scheme. This follows from our general result showing that CMT_k security implies $\mathsf{CDY}_\mathsf{a}^*$ and $\mathsf{CDY}_\mathsf{n}^*$ (Corollary 3). But this does not imply the ability to build restrictive $\mathsf{CMT}_\mathsf{k}^*$, $\mathsf{CMT}_\mathsf{n}^*$, or $\mathsf{CMT}_\mathsf{a}^*$ attacks that require the non-adversarially controlled parts of the two decryption contexts to be identical (see Theorem 10 in the full version [126].)

Prior work has provided (in our terminology) $\mathsf{CMT}_\mathsf{k}^*$ attacks for GCM [87, 71], AES-GCM-SIV [157, 119], ChaCha20/Poly1305 [87, 119], XChaCha20/Poly1305 [119], and OCB3 [3]. An open question of practical interest [158] is whether there is a $\mathsf{CMT}_\mathsf{k}^*$ attack against SIV. We resolve this open question, showing an attack that works in time about $2^{n/3}$. It requires new techniques related to the fast solution of k-sum problems, as we explain below.

Attack on 1-block SIV We consider SIV over an n-bit block cipher (for $n \geq 64$) as defined in the draft NIST specification [150]. For ease of exposition, we restrict to the case of an n-bit message and no associated data, and describe how to generalize this to the multi-block case in Appendix D in the full version [126]. Pseudocode for the scheme with these parameter choices is given in Figure 2.10.

Here, the $\mathrm{CMT}^*_{\mathsf{k}}$ adversary seeks to produce a ciphertext $C = C_1 \parallel \mathsf{tag}$ and two $2n$-bit keys $K = K_1 \parallel K_2$ and $K' = K'_1 \parallel K'_2$ such that $\mathsf{SIV\text{-}Decrypt}(K, C) \neq \perp$ and $\mathsf{SIV\text{-}Decrypt}(K', C) \neq \perp$. Notice from Figure 2.10 that this reduces to two simultaneous IV checks passing which can be written as

$$\mathsf{tag} = \mathsf{CMAC}^*(K_1, C_1 \oplus E_{K_2}(\mathsf{tag}\ \&\ \mathsf{c})) = \mathsf{CMAC}^*(K'_1, C_1 \oplus E_{K'_2}(\mathsf{tag}\ \&\ \mathsf{c}))$$

where $\mathsf{c} = 1^{n-64}01^{31}01^{31}$ is a constant specified by the SIV standard. Our attack strategy will be to choose tag arbitrarily, so we can treat this as a constant value. Towards solving for the remaining variable C_1, we can substitute in the definition of CMAC^* to get

$$\begin{aligned}\mathsf{tag} &= E_{K_1}((2 \cdot E_{K_1}(0^n)) \oplus E_{K_1}(2 \cdot E_{K_1}(0^n)) \oplus C_1 \oplus E_{K_2}(\mathsf{tag}\ \&\ \mathsf{c})) \\ &= E_{K'_1}((2 \cdot E_{K'_1}(0^n)) \oplus E_{K'_1}(2 \cdot E_{K'_1}(0^n)) \oplus C_1 \oplus E_{K'_2}(\mathsf{tag}\ \&\ \mathsf{c}))\,,\end{aligned}$$

which we can rearrange the two equalities by solving for the variable C_1, giving us the following:

$$\begin{aligned}C_1 &= E^{-1}_{K_1}(\mathsf{tag}) \oplus (2 \cdot E_{K_1}(0^n)) \oplus E_{K_1}(2 \cdot E_{K_1}(0^n)) \oplus E_{K_2}(\mathsf{tag}\ \&\ \mathsf{c}) \\ &= E^{-1}_{K'_1}(\mathsf{tag}) \oplus (2 \cdot E_{K'_1}(0^n)) \oplus E_{K'_1}(2 \cdot E_{K'_1}(0^n)) \oplus E_{K'_2}(\mathsf{tag}\ \&\ \mathsf{c})\,. \end{aligned} \tag{2.5}$$

The above implies that it suffices now to find K_1, K_2, K'_1, K'_2 that satisfy Equation 2.5.

To ease notation, we define four helper functions, one for each term:

$$\begin{aligned}
F_1(K_1) &:= E^{-1}_{K_1}(\mathsf{tag}) \oplus 2 \cdot E_{K_1}(0^n) \oplus E_{K_1}(2 \cdot E_{K_1}(0^n)), \\
F_2(K_2) &:= E_{K_2}(\mathsf{tag} \mathbin{\&} \mathsf{c}), \\
F_3(K_1) &:= E^{-1}_{K_1'}(\mathsf{tag}) \oplus 2 \cdot E_{K_1'}(0^n) \oplus E_{K_1'}(2 \cdot E_{K_1}(0^n)), \\
F_4(K_2') &:= E_{K_2'}(\mathsf{tag} \mathbin{\&} \mathsf{c}),
\end{aligned}$$

and recast Equation 2.5 as a 4-sum problem

$$F_1(K_1) \oplus F_2(K_2) \oplus F_3(K_1') \oplus F_4(K_2') = 0 .$$

If these were independent random functions, then we could directly apply Wagner's k-tree algorithm [164] for finding a 4-way collision (also referred to as the generalized birthday problem). But even modeling E as an ideal cipher, the functions are neither random nor independent. For example, $F_1(x) = F_3(x)$ always.

Towards resolving this, we first ensure that the keys K_1, K_2, K_1', and K_2' are domain separated. This can be easily arranged: see Figure 2.11 for the pseudocode of our $\mathrm{CMT}^*_{\mathsf{k}}$ adversary $\mathcal{A}$ against SIV. We now turn to lower bounding $\mathcal{A}$'s advantage, which consists of two primary steps.

The first is that we argue that, in $\mathrm{CMT}^*_{\mathsf{k}}$ when running our adversary against SIV, the helper-function outputs are statistically close to uniform. Then, we show that Wagner's approach works for such values.

We observe that F_2 and F_4 trivially behave as independent random functions in the ideal cipher model for E. The analysis for F_1 and F_3 is more involved. We use the following lemma, which bounds the distinguishing advantage between a uniform n-bit string and the output of a query to either F_1 or F_3.

```
$\underline{\mathcal{A}()}$:
$\mathsf{c} \leftarrow 1^{n-64}01^{31}01^{31}$
// Arbitrarily pick a tag
$\mathsf{tag} \leftarrow_{\$} \{0,1\}^n \setminus \{0^n\}$
// Define helper functions
Def $F_1(K_1) \leftarrow E^{-1}_{K_1}(\mathsf{tag}) \oplus 2 \cdot E_{K_1}(0^n) \oplus E_{K_1}(2 \cdot E_{K_1}(0^n))$
Def $F_2(K_2) \leftarrow E_{K_2}(\mathsf{tag}\ \&\ \mathsf{c})$
Def $F_3(K_1) \leftarrow E^{-1}_{K_1'}(\mathsf{tag}) \oplus 2 \cdot E_{K_1'}(0^n) \oplus E_{K_1'}(2 \cdot E_{K_1}(0^n))$
Def $F_4(K_2') \leftarrow E_{K_2'}(\mathsf{tag}\ \&\ \mathsf{c})$
// Generate lists
For $i = 1, ..., q$:
    $x \leftarrow \mathsf{encode}_{128-2}(i)$
      // Domain separate the keys
    $K_1 \leftarrow \mathtt{00} \,\|\, x;\ K_2 \leftarrow \mathtt{01} \,\|\, x;\ K_1' \leftarrow \mathtt{10} \,\|\, x;\ K_2' \leftarrow \mathtt{11} \,\|\, x$
      // Query a row
    $L_1[i] \leftarrow F_1(K_1);\ L_2[i] \leftarrow F_2(K_2);\ L_3[i] \leftarrow F_3(K_1');\ L_4[i] \leftarrow F_4(K_2')$
// Find an 4-way collision using Wagner's k-tree algorithm [164]
res $\leftarrow \mathcal{A}.\mathsf{fourWayCollision}(L_1, L_2, L_3, L_4)$
If res $= \emptyset$:
    Return $\perp$
// Repackage the collision into ciphertext and keys
$(x_1, x_2, x_3, x_4) \leftarrow$ res
$C_1 \leftarrow F_1(x_1) \oplus F_2(x_2)$
$K_1 \leftarrow \mathtt{00} \,\|\, x_1;\ K_2 \leftarrow \mathtt{01} \,\|\, x_2;\ K_1' \leftarrow \mathtt{10} \,\|\, x_3;\ K_2' \leftarrow \mathtt{11} \,\|\, x_4$
Return $C_1 \,\|\, \mathsf{tag}, K_1 \,\|\, K_2, K_1' \,\|\, K_2'$
```

Figure 2.11: Pseudocode for $\mathrm{CMT}^*_{\mathsf{k}}$ attack on SIV-1b, where fourWayCollision is defined in Appendix C in the full version [126].

Lemma 4. *Let* $\mathsf{tag} \in \{0,1\}^n \setminus \{0^n\}$ *and* σ *be an* n*-bit random permutation with inverse* σ^{-1} *and* U *be the uniform random variable over* n *bit strings. Define* n*-bit random variables (over the choice of* σ*)*

$$A := \sigma^{-1}(\mathsf{tag})\,, \qquad B := 2 \cdot \sigma(0^n)\,, \qquad C := \sigma(2 \cdot \sigma(0^n))\,,$$

where $\cdot$ *denotes multiplication in* $\mathrm{GF}(2^n)$*. Then no adversary that makes one query to a procedure* P *can distinguish between* $P \mapsto (U, U, U)$ *and* $P \mapsto (A, B, C)$ *with probability greater than* $6 \cdot 2^{-n}$*.*

The proof proceeds by constructing identical-until-bad games and applying the *fundamental lemma of game playing* [26] to discern the distinguishing advantage. The proof appears in Appendix C in the full version [126].

We combine this with the following technical statement about applying Wagner's k-tree algorithm [164] to almost-random lists.

Theorem 5. *Let L be a list of ℓ 4-tuples $x = (x_1, x_2, x_3, x_4)$, where each entry x is distinguishable from an 4-tuple of independent uniformly random values with probability at most ξ. Let L_1, L_2, L_3, and L_4 be lists of 1-index (x_1), 2-index (x_2), 3-index (x_3), and 4-index (x_4) elements of L respectively. Then Wagner's k-tree algorithm [164] finds a solution $(y_1, y_2, y_3, y_4) \in L_1 \times L_2 \times L_3 \times L_4$ such that*

$$y_1 \oplus y_2 \oplus y_3 \oplus y_4 = 0\,,$$

with probability at least

$$(1 - \ell \cdot \xi)\left(1 - \exp\left(-\frac{\ell^2 \cdot 2^{-n/3}}{8}\right)\right)\left(1 - \exp\left(1 - \frac{\ell^4 \cdot 2^{-4n/3}}{8} - \frac{2}{\ell^4 \cdot 2^{-4n/3}}\right)\right),$$

and time at most

$$20\ell + 4\ell^2 \cdot 2^{-n/3} + 4\mathsf{Sort}(\ell) + 2\mathsf{Sort}((1/2)\ell^2 \cdot 2^{-n/3})\,,$$

where $\mathsf{Sort}(k)$ denotes the time to sort a list of k items.

The proof proceeds by analyzing the algorithm step-by-step and at each step applying Chernoff bounds [83] to compute a lower bound on the success probability. The proof appears in Appendix C in the full version [126].

With Lemma 4 and Theorem 5, we can now prove a lower bound on the advantage of the $\mathrm{CMT}^*_{\mathsf{k}}$ adversary in Figure 2.11.

Theorem 6. *Let $\mathcal{A}$ be the $\mathrm{CMT}^*_{\mathsf{k}}$ adversary against SIV over an n-bit ideal cipher E,* *detailed in Figure 2.11. It makes $10q$ queries to E and takes at most*

$$35q + 4q^2 \cdot 2^{-n/3} + 4\mathsf{Sort}(q) + 2\mathsf{Sort}((1/2)q^2 \cdot 2^{-n/3}) + 11\,,$$

time, where $\mathsf{Sort}(k)$ *is the cost of sorting a list of* k *items. Then the advantage*

$$\begin{aligned}\mathbf{Adv}_{\mathrm{SIV}}^{\mathrm{CMT}_k^*}(\mathcal{A}) \geq (1 - 8q \cdot 2^{-n}) &\left(1 - \exp\left(-\frac{q^2 \cdot 2^{-n/3}}{8}\right)\right) \\ &\left(1 - \exp\left(1 - \frac{q^4 \cdot 2^{-4n/3}}{8} - \frac{2}{q^4 \cdot 2^{-4n/3}}\right)\right).\end{aligned} \tag{2.6}$$

Proof. By construction, the adversary $\mathcal{A}$ (Figure 2.11) wins whenever it finds a collision, so it suffices to lower bound this probability. First, the domain separation over the keys ensures that the two helper functions never query the ideal cipher with the same key. This, by the properties of the ideal cipher, ensures independence of the outputs. Second, F_2 and F_4 call the ideal cipher only once on a fixed output under a new key each invocation, so their outputs are indistinguishable from an n-bit uniform random value. Third, F_1 and F_3 call the ideal cipher three times under the same key each invocation. However, applying Lemma 4 gives us that their outputs are distinguishable from an n-bit uniform random value with probability at most $6 \cdot 2^{-n}$. So, by the union bound, a row of outputs $(F_1(K_1), F_2(K_2), F_3(K_1'), F_4(K_2'))$ is distinguishable from four independent, uniformly random outputs with probability at most $8 \cdot 2^{-n}$. Then, Theorem 5 tells us that the function $\mathsf{fourWayCollision}$ called by $\mathcal{A}$ finds a collision with probability at least that of Equation 2.6.

It remains to analyze the cost of the adversary $\mathcal{A}$. First, it costs 2 operations to initialize c and tag. Second, since each loop iteration costs 15 operations, the loop costs $15q$ operations. Third, from Theorem 5, finding a 4-way collision on four lists of size q using Wagner's k-tree algorithm [164] costs at most

$$20q + 4q^2 \cdot 2^{-n/3} + 4\mathsf{Sort}(q) + 2\mathsf{Sort}((1/2)q^2 \cdot 2^{-n/3})$$

operations. Fourth, repackaging the collision and returning costs 9 operations. So, the runtime is at most

$$35q + 4q^2 \cdot 2^{-n/3} + 4\mathsf{Sort}(q) + 2\mathsf{Sort}((1/2)q^2 \cdot 2^{-n/3}) + 11\,.$$

Finally, since each loop iteration makes 10 ideal cipher queries, the algorithm makes $10q$ queries. □

In the following corollary, we show that when the adversary makes approximately $2^{n/3}$ queries, it can win $\mathrm{CMT}_{\mathsf{k}}^{*}$ against SIV with high probability, taking time approximately $2^{n/3}$.

Corollary 7. *Let $\mathcal{A}$ be the $\mathrm{CMT}_{\mathsf{k}}^{*}$ adversary against SIV over an n-bit ideal cipher E, detailed in Figure 2.11 with $q = 10 \cdot 2^{n/3}$. It makes $100 \cdot 2^{n/3}$ queries to E and takes at most*

$$750 \cdot 2^{n/3} + 4\mathsf{Sort}(10 \cdot 2^{n/3}) + 2\mathsf{Sort}(50 \cdot 2^{n/3}) + 11\,,$$

time, where $\mathsf{Sort}(n)$ is the cost of sorting a list of n items. Then

$$\mathbf{Adv}_{\mathrm{SIV}}^{\mathrm{CMT}_{\mathsf{k}}^{*}}(\mathcal{A}) \geq \left(1 - 80 \cdot 2^{-2n/3}\right)\left(1 - \exp\left(-12.5 \cdot 2^{n/3}\right)\right)\left(1 - \exp\left(-1249\right)\right).$$

2.5 Related Work

Key commitment for authenticated encryption was introduced in Farshim, Orlandi, and Rosie [78] through full robustness (FROB), which in turn was inspired by key robustness notions in the public key setting by Abdalla, Bellare, and Neven [1] and refined by Farshim et al. [80]. The FROB game asks that a ciphertext only be able to decrypt under a single key. However, the FROB game was defined for randomized authenticated encryption. Grubbs, Lu, and Ristenpart [87] adapted the FROB game to work with associated data, where they ask that a ciphertext only be able to decrypt under a single key (with no constraints on the associated data.) This notion was further generalized by Bellare and Hoang [16] to the nonce-based setting, with their committing security 1 (CMT-1) definition. The CMT-1 game asks that a ciphertext only be able to decrypt under a single key (with no constraints on the nonce nor the associated data.)

The real-world security implications of key commitment were first highlighted by Dodis et al. [71] where they exploited the lack of key commitment when encrypting attachments in Facebook Messenger's message franking protocol [77] to send abusive images that cannot be reported. Albertini et al. [3] generalized this attack from images to other file formats and called attention to more settings where lack of key commitment can be exploited to defeat integrity. While both these attacks targeted integrity, Len, Grubbs, and Ristenpart [119] introduced partitioning oracle attacks and showed how to use them for password guessing attacks by exploiting lack of key commitment to obtain large speedups over standard dictionary attacks, endangering confidentiality.

Proposals for constructing key committing ciphers also started in the Farshim, Orlandi, and Rosie paper [78] where they showed that single-key Encrypt-then-MAC, Encrypt-and-MAC, and MAC-then-Encrypt constructions produce key committing ciphers, when the MAC is collision-resistant. Grubbs, Lu, and Ristenpart [87] showed that the Encode-then-Encipher construction [23] was key committing. Dodis et al. [71] proposed a faster compression function-based key committing AEAD construction termed *encryptment*, and also discussed the closely related Duplex construction [36], which is also key committing. Albertini et al. [3] formally analyzed the folklore padding zeroes and key hashing transforms and showed that they produce key committing AEAD at a lower performance cost than prior constructions. Bellare and Hoang [16] constructed key committing variants of GCM and GCM-SIV termed CAU-C1 and CAU-

SIV-C1, and generic transforms UtC and RtC that can be used to turn unique-nonce secure and nonce-reuse secure AEAD schemes respectively into key committing AEAD schemes.

The potential risk of delegating authenticity of an AEAD entirely to a non-collision-resistant MAC is folklore. Farshim, Orlandi, and Rosie [78] who introduced the notion of committing AEAD also cautioned against using non-collision-resistant MACs and CBC-MAC in particular.

On February 7, 2023, NIST announced the selection of the Ascon family for lightweight cryptography standardization [159]. The finalist version of Ascon [69] specifies two AEAD parameter sets Ascon-128 and Ascon-128a. Both parameter sets specify a 128 bit tag, which by the birthday bound, upper bounds the committing security at 64 bits. But, since the underlying algorithm is a variant of the Duplex construction with a 320-bit permutation, and the same specification specifies parameters for a hash function with 128-bit collision resistance, one can specify an AEAD with 128-bit committing security by tweaking parameters.

Concurrent work. In independent and concurrent work made public very recently, Chan and Rogaway [51] introduced a new definitional framework for committing AE. Their goal is to capture multiple different types of commitment attacks—what they call *misattributions*, or an adversary being able to construct distinct pairs (K, N, A, M) and (K', N', A', M') that both “explain” a single ciphertext C—in a unified way. Their main definition only captures commitment to an entire (K, N, A, M) tuple; but in [51, Appendix A], they briefly describe an extension to only require commitments to a subset of the values.

The extended version of their framework is similar to our CMT[Σ] definition.

While both frameworks aim to capture granular win conditions beyond CMT-3, they are orthogonal. Their framework models the multi-key setting with many randomly chosen unknown-to-the-adversary, known-to-the-adversary, and chosen-by-the-adversary keys. While our CMT[Σ] captures the distinction between *permissive* and *restrictive* notions, and settings that impose restrictions on the nonce and associated data. We also introduce the notion of context discoverability and describe its relation to CMT[Σ].

Chan and Rogaway [51] also independently observed that AEAD with non-preimage resistant MACs are vulnerable to commitment attacks and show attacks on GCM and OCB3 similar to the ones we give in Section 2.3.

CHAPTER 3

THE OCH AUTHENTICATED ENCRYPTION SCHEME*

Recall that in an authenticated encryption with associated data (AEAD) scheme [144], encryption takes a key, nonce, associated data, and message to deterministically return a ciphertext. The classical security requirement is unique-nonce AE (NAE) security. This means privacy of the message, and authenticity of the message and associated data, assuming encryption never reuses a nonce. These were the security and functionality goals for which the most widely used AEAD schemes were designed over two decades ago, including AES-GCM [124] and ChaCha20-Poly1305 [136].

Practitioners and researchers have in the last few years reached consensus that NAE, and the widely used schemes that target it, suffer from a variety of critical limitations. First, NAE does not guarantee what has come to be called key commitment: an adversary can construct a ciphertext so that it is decryptable by multiple maliciously chosen keys. A spate of work [118, 2, 125, 71, 86, 157, 100] highlights how lack of key commitment leads to attacks in important applications. A stronger goal that implies key commitment is context commitment (CMT) [17]: a ciphertext can only be decrypted under a particular key, nonce, and associated data triple. Second, the concrete security of in-use schemes limits the number of bytes that can be encrypted under any given key, forcing complicated key rotation logic when used at the scale of modern workloads [107]. Finally, widely used schemes do not support nonce hiding [22]. Nonce hiding is important when nonces contain sensitive information such as device identifiers, and also fills a gap in classical NAE. (The latter actually fails to hide the message for certain choices of public nonces [22].)

*This chapter is joint work with Mihir Bellare, Viet Tung Hoang, Julia Len, and Thomas Ristenpart. The proceedings version of this paper will appear at CCS 2025 [128], and an earlier version was presented at Real World Crypto 2024 [32]. This chapter is lightly edited from the full version we are preparing for public release.

The academic literature includes techniques for rectifying each of these problems in isolation, but no AEAD schemes have been proposed that achieve *all* of them in a high-performance way.

Our contributions. We set out to build AEAD schemes that could replace today's NAE schemes and protect data for decades to come. To do so we provide a new, clean-slate approach to secure, efficient AEAD schemes and apply it to build OCH (offset codebook with hashing): a permutation-based AEAD. OCH is the first scheme to provide the following combination of features:

- *128-bit NAE security:* OCH's formal confidentiality and authenticity guarantees scale to even settings where adversaries have unrealistically large resources, having to get close to 2^{128} queries to have a meaningful attack.
- *128-bit CMT security:* OCH is secure against the strongest form of commitment-style attacks, scaling up to adversaries that can perform close to 2^{128} computations.
- *Nonce versatility and hiding:* OCH can be used with random nonces of ≥ 192 bits to avoid collisions, or structured nonces such as counters. It can hide nonces.
- *Speed:* OCH is fast, maximally parallelizable, and utilizes CPU pipelines to achieve high throughput. For example, OCH easily achieves > 1 GB/s throughput on a variety of platforms. It is almost as fast as AES-GCM and OCB [112, 151]—the fastest NAE schemes on many platforms—despite those schemes not meeting any of the above security goals.

See Figure 3.1 for a comparison of OCH and prior schemes.

OCH relies on a large number of design ideas, many of which are new to this work and of broader applicability. We summarize OCH's design at a high level, highlighting novel components and starting from the bottom. We build a tweakable Even-Mansour

Scheme	**128-bit NAE**	**128-bit CMT**	**192-bit Nonce**	**Nonce privacy**	**1GB/s encrypt**
AES128-GCM [124, 73]	✗	✗	✗	✗	✓
AES256-GCM [124, 73]	✗	✗	✗	✗	✓
ChaCha20/Poly1305 [137]	✓	✗	✗	✗	✓
CommitKey[GCM256] [90]		✗	✓	✗	✓
XChaCha20/Poly1305 [9]	✓	✗	✓	✗	✓
AES-256-GEM [8]		✗	✓	✗	✓
XAES-256-GCM [163]		✗	✓	✗	✓
DNDK-GCM256 [89]		✗	✓	✗	✓
CAU-C1 [15]		✗	✗	✗	✓
CTX[GCM256] [52]		✓		✗	✓
Ascon-AEAD128 [161]	✗	✗	✗	✗	✗
Aegis-256 [167, 65]		✗	✓	✗	✓
Deoxys-I-256 [104]		✗	✗	✗	✓
TurboSHAKE128 [60]	✓	✓	✓	✗	✗
AreionOCH (§3.4)	✓	✓	✓	✓	✓

Figure 3.1: Comparison of widely used AEAD schemes (top group), transforms of existing schemes (middle), and clean-state designs (bottom). NAE and CMT refer to multi-user nonce-based AE security and context commitment security, respectively. The last column lists whether the scheme can encrypt a billion bytes per second, for 1024 byte messages with 13 bytes of associated data (corresponding to TLS) on an Intel Raptor Lake CPU.

blockcipher [106, 116, 56] from an underlying cryptographic permutation; the construction, however, is not a generic application of prior work but customized to enable fast amortized generation of offsets for use in a new ΘCB-like [112] authenticated encryption (AE) that: (1) supports nonce-hiding, (2) omits support for associated data, and (3) does *not* provide authenticity. Instead, it meets a new, weaker-than-NAE security goal for authenticated encryption (without associated data) called NAX. Intuitively, tags need not be unforgeable but just be computationally almost-xor-universal.

We then provide a new generic transform called Xor-then-Hash (XtH) that uses a collision-resistant PRF to convert a (nonce-hiding) NAX-secure AE scheme into a (nonce-hiding) context-committing AEAD scheme. We build the CR-PRF from a permutation using a Sponge construction [39, 37]. XtH can also be applied to standard AEAD schemes, in which case it can be viewed as a new generic transform for rendering AEAD context committing, and is more efficient than the prior CTY [17] and CTX [52] transforms, particularly for the short associated data strings most often used in practice.

The final component is a specialized mode for handling very short messages. Here, we provide a new, nonce-hiding variant of the synthetic IV mode [149] that can be built from the same tweakable blockcipher as ΘCX.

We provide a formal analysis of all of the above. To make it both tractable and easier to verify, our analysis is highly modular. This also means that our intermediate design components can be more readily used elsewhere with formal analyses already provided.

Limitations. Non-goals for OCH are achieving nonce-misuse-resistance [149] or robust AEAD security [95]. These provide improved confidentiality and authenticity in the face of practically relevant threats, such as accidental nonce reuse or accidental release of unverified plaintext. But achieving them requires more than one cryptographic pass over plaintexts, and would bar the ability to achieve performance competitive with one-pass NAE schemes.

Summary. We designed the first AEAD scheme that simultaneously provides high-security parameter NAE security, context commitment security, and support for nonce hiding, while performing almost as well as insufficiently secure schemes such as AES-GCM and ChaCha20/Poly1305. Our implementation includes optimizations for various platforms. We will be open-sourcing and making it public before publication of this paper.

3.1 Background and Related Work

We provide a high-level overview of modern design goals, interleaving discussion of related work.

High security scaling. We desire schemes that have good (multi-user) security bounds, so that security holds even when used across many instances and for today's large-scale workloads. For example, AES-based schemes with birthday bound security fail at around 2^{64} AES invocations, which limits scalability. TLS 1.3 does not allow encryption of more than $2^{24.5}$ records under a single key due to this birthday bound issue [141, §5.5]. See [107] for a more detailed discussion of modern scaling challenges. By providing significantly higher security, we can instead avoid hard usage limits in practice. Thus, we have the following goal:

128-bit NAE: *AEAD should achieve 128-bit NAE security, meaning breaking confidentiality and authenticity security would require close to an intractably large* 2^{128} *adversarial queries.*

In theory, it is not hard to build AEAD that achieves 128-bit NAE. and in fact the academic literature has a surfeit of options for building AEAD schemes. For example, one might build an Encrypt-then-MAC AEAD using Rijndael-256 in counter mode for encryption, together with HMAC-SHA256 for message authentication. But this will be a lot slower than what we achieve. We further review the landscape of approaches for building AEAD in Section 3.7.

Context commitment. NAE security does *not* guarantee what has come to be called key commitment, as first formalized by [79]: an adversary can construct a ciphertext so that it is decryptable by multiple maliciously chosen keys. An increasingly large body of work shows how lack of key commitment leads to attacks in practical settings including: abuse reporting in encrypted messaging [71, 86], password-based encryption [118], password-based key exchange [118], anonymous public key encryption [118], key rotation schemes [2], and symmetric hybrid (also called envelope) encryption [2]. Subsequent works introduced security goals stronger than key commitment, such as context commitment [15, 125, 52]. It requires that ciphertexts must only be decryptable under a single key, nonce, and associated data triple—what we refer to as a context, following [125]. Attacks have shown that even schemes that achieve key commitment can fail to meet this stronger goal [125]. We refer to context commitment as CMT.

Given all the security issues stemming from lack of commitment, the academic and practitioner consensus (c.f., [107, 55, 32, 31, 118, 140, 18]) is that general-purpose AEAD should achieve at least key commitment and, ideally, context commitment as well.

Prior academic work, as well as deployed implementations, suggest transforms that convert an existing non-CMT AEAD to be CMT [2, 107, 72, 64, 15, 52, 90]. But these transforms don't address other deficiencies in the underlying AEAD schemes, such as not achieving 128-bit NAE. One could apply known frameworks for building new schemes that are 128-bit NAE (such as Encrypt-then-MAC, as mentioned above), and then apply the transforms. But, again, these won't be as fast as our approach.

Ascon [68, 161], the winner of a recent lightweight AEAD competition, appears to achieve CMT security, but only against adversaries limited to much less than 2^{64} computations. Other schemes that achieve 128-bit CMT and 128-bit NAE are the recent SHAKE and TurboSHAKE schemes [60] built from a cryptographic hash function. While elegant and fast on short messages, these schemes are inherently serial and not as fast as ours for larger messages.

128-bit CMT: *AEAD should achieve 128-bit CMT security, meaning an adversary must use* 2^{128} *computations to find a ciphertext that decrypts under two different contexts.*

Large nonces and nonce hiding. Finally, we want to support both stateful and randomized nonces, or nonces that are some combination of the two. Here there are two issues. First is that nonces for widely used schemes are too small to allow encrypting, with the same key, a large number of messages with random nonces. For example, AES-GCM's maximum effective nonce length is 96 bits (longer nonces get hashed to 96 bits), and so one cannot use random nonces to process more than 2^{32} messages before security breaks down. But large scale deployments may encrypt that many messages in a few seconds, forcing more complicated software engineering to work around the limitations of AES-GCM (frequent rekeying, state management for counter-based nonces, etc.).

A second issue is that, as Bellare, Ng, and Tackmann [22] observed, publicly transmitted non-random nonces can be privacy damaging, such as leaking the number of encryptions performed, the identity of devices performing encryptions, or even partial information about messages for some ways of choosing nonces. While some prior work has given schemes that achieve nonce-hiding [22], no schemes widely used in practice provide it. Ideally, a scheme would be versatile, allowing sufficiently large random nonces ($\geq$ 192 bits to avoid collisions) that need not be hidden, stateful or structured nonces that are kept secret, or nonces that combine both. We refer to this as supporting versatile nonces.

Versatile nonces: *AEAD should support random nonces of sufficient size (≥ 192 bits), and allow hiding part or all of a nonce.*

3.2 Preliminaries

Notation. We let $\mathbb{N}_0$ be the set of non-negative integers. We refer to elements of $\{0,1\}^*$ as *bitstrings*, denote the length of a bitstring x by $|x|$ and refer to bitstrings of length n as n-bit strings. We denote the empty string with ε, the n-bit all-zero and all-one strings by 0^n and 1^n, the concatenation of two bitstrings $x, y \in \{0,1\}^*$ by $x \| y$, and the bitwise xor of two n-bit strings $x, y \in \{0,1\}^n$ by $x \oplus y$. Given an n-bit string $x = x_{n-1} \cdots x_1 x_0 \in \{0,1\}^n$, we define its most significant bit $\mathsf{msb}(x) := x_{n-1}$, its logical left shift $(x \ll i) := x_{n-1-i} \cdots x_0 0^i$ where $i < n$, and its number representation $\mathsf{str2num}_n(x) := \sum_{i=0}^{n-1} x_i 2^i$. We note that $\mathsf{str2num}_n$ is invertible and denote its inverse $\mathsf{num2str}_n$, which maps all numbers $0 \leq A < 2^n$ to n-bit strings such that $\mathsf{str2num}_n(\mathsf{num2str}_n(A)) = A$. An n-tuple $(x_1, \ldots, x_n) \in (\{0,1\}^*)^n$ of bitstrings is a sequence of n bitstrings, and we denote the set of all bitstring tuples with $\{0,1\}^{**}$.

All the sets we consider in this work are finite. For a finite set X, we use $x \leftarrow_\$ X$ to denote sampling a uniform, random element from X and assigning it to x.

We use *code-based games* [27, 134] for our definitions and proofs. A *procedure* P is a sequence of code-like statements. We assume all the variables in a procedure are initialized, and all the inputs and outputs are in the specified domain and range, respectively. We use the symbol $\perp$ to denote errors, and assume it is always distinguishable. For a procedure P, we let $y \leftarrow_\$ P^{O_1,\cdots}(x_1, \ldots)$ denote running P on inputs $x_1, \ldots$, with oracle access to $O_1, \ldots$, and assigning the output to y. We use $(P \Rightarrow x)$ to denote the event that procedure P outputs x, over the coins of P.

Permutations. A w-bit permutation is function $\Pi : \{0,1\}^w \to \{0,1\}^w$, with an inverse $\Pi^{-1} : \{0,1\}^w \to \{0,1\}^w$. Let $\mathrm{Perm}(\mathcal{M})$ be the set of all permutations on $\mathcal{M}$, and let $\widetilde{\mathrm{Perm}}(\mathcal{T}, \mathcal{M})$ be the set of all permutations on $\mathcal{M}$ tweaked by $\mathcal{T}$, that is, the set of all families of permutations on $\mathcal{M}$ indexed by $T \in \mathcal{T}$.

Ideal permutation model. In this paper, we design schemes based on an underlying permutation Π. To analyze them, we use the *ideal permutation model* [7, 105, 63] where we model this permutation as a uniformly random permutation $\Pi \leftarrow_{\$} \mathrm{Perms}(\mathcal{M})$ over the message space $\mathcal{M}$. In this model, the adversary is additionally given access to this permutation and its inverse with the oracles Perm and PermInv, and probabilities are additionally taken over the random choice of the underlying permutation Π. We assume this underlying permutation is public and provide the adversary access to the inverse oracle PermInv even if the construction does not use the inverse.

Multi-user security. We consider security in the *multi-user or multi-key setting* [132, 30, 122] where the adversary can query oracles under different keys $K_i \leftarrow_{\$} \mathcal{K}$ chosen independently and uniformly at random from the keyspace $\mathcal{K}$. and the adversary wins if it compromises the security of *any one* of those keys. This captures settings like TLS, Signal, and server key-rotation where compromising one key is sufficient to cause harm. Following convention, for the remainder of this paper, we will use term "multi-user" but note that it is misleading since users routinely have multiple keys.

Tweakable blockciphers. A tweakable blockcipher (TBC) [120] is a function $\tilde{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{M}$, defined over a key space $\mathcal{K} \subset \{0,1\}^*$, tweak space $\mathcal{T} \subset \{0,1\}^*$,

```
Game $\mathrm{TPRP}^{\mathrm{real}}_{\tilde{E}}(\mathcal{A})$:
$\Pi \leftarrow\!\!\$\ \mathrm{Perm}(\mathcal{M})$
$b \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{TPRP}\{\mathrm{Kg,\ Enc,\ Dec}\},\Pi^{\pm}}$
return $b$

$\mathrm{TprpKg}()$:
    $u \leftarrow u+1$;  $K_u \leftarrow\!\!\$\ \mathcal{K}$

$\mathrm{TprpEnc}^{\Pi^{\pm}}(i, T, X)$:
    return $\tilde{E}^{p\pm}_{K_i}(T, X)$

$\mathrm{TprpDec}^{\Pi^{\pm}}(i, T, Y)$:
    return $\tilde{D}^{p\pm}_{K_i}(T, Y)$
```

```
Game $\mathrm{TPRP}^{\mathrm{rand}}_{\tilde{E}}(\mathcal{A})$:
$\Pi \leftarrow\!\!\$\ \mathrm{Perm}(\mathcal{M})$
$b \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{TPRP}\{\mathrm{Kg,\ Enc,\ Dec}\},\Pi^{\pm}}$
return $b$

$\mathrm{TprpKg}()$:
    $u \leftarrow u+1$;  $\tilde{\pi}_u \leftarrow\!\!\$\ \widetilde{\mathrm{Perm}}(\mathcal{T}, \mathcal{M})$

$\mathrm{TprpEnc}(i, T, X)$:
    return $\tilde{\pi}_i(T, X)$

$\mathrm{TprpDec}(i, T, Y)$:
    return $\tilde{\pi}_i^{-1}(T, Y)$
```

Figure 3.2: Games ($\mathrm{TPRP}^{\mathrm{real}}, \mathrm{TPRP}^{\mathrm{rand}}$) for a TBC $\tilde{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{M}$, based on a permutation $\Pi : \mathcal{M} \to \mathcal{M}$. If we set the tweakspace $\mathcal{T} = \{\varepsilon\}$, these games reduce to ($\mathrm{SPRP}^{\mathrm{real}}, \mathrm{SPRP}^{\mathrm{rand}}$) for a blockcipher $E : \mathcal{K} \times \mathcal{M} \to \mathcal{M}$.

and input space $\mathcal{M} \subset \{0,1\}^*$, where for each key $K \in \mathcal{K}$ and each tweak $T \in \mathcal{T}$, the function $\tilde{E}^T_K(\cdot) := \tilde{E}(K, T, \cdot)$ is a permutation on $\mathcal{M}$, and we denote its inverse with $\tilde{D}^T_K(\cdot) := \tilde{D}(K, T, \cdot)$. We require keys $|K| =: k$ and inputs $|M| =: n$ to be fixed length, but allow structured tweaks. We recover traditional blockciphers by setting $\mathcal{T} = \{\varepsilon\}$.

In this work, we analyze tweakable blockciphers $\tilde{E}$ built from a public permutation Π, and consider their security in the *ideal permutation model*, where we model the permutation Π as a perfectly random permutation $\Pi \leftarrow\!\!\$\ \mathrm{Perm}(\mathcal{M})$.

For a TBC $\tilde{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{M}$ based on a permutation $\Pi \leftarrow\!\!\$\ \mathrm{Perm}(\mathcal{M})$ and an adversary $\mathcal{A}$ we define multi-user (strong) TPRP security as $\mathsf{Adv}^{\mathrm{tprp}}_{\tilde{E}}(\mathcal{A}) := \Pr[\mathrm{TPRP}^{\mathrm{real}}(\mathcal{A}) \Rightarrow 1] - \Pr[\mathrm{TPRP}^{\mathrm{rand}}(\mathcal{A}) \Rightarrow 1]$, using a pair of games $\mathrm{TPRP}^{\mathrm{real}}$ and $\mathrm{TPRP}^{\mathrm{rand}}$ defined in Figure 3.2.

PRFs and CR-PRFs. A variable-output-length pseudorandom function (PRF) is a function $\mathsf{F} : \mathcal{K} \times \mathcal{L} \times \mathcal{X} \to \mathcal{Y}$, defined over a key space $\mathcal{K} \subseteq \{0,1\}^*$, output length space $\mathcal{L} \subseteq \mathbb{N}$, input space $\mathcal{X} \subseteq \{0,1\}^{**}$, and output space $\mathcal{Y} \subseteq \{0,1\}^*$. We allow the input space to include tuples, require the lengths of keys $|K| =: k$ be a public constant, and require

```
Game $\mathrm{PRF}^{\mathrm{real}}_{\mathsf{F}}(\mathcal{A})$
$b \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Kg},\textsc{Func}}$
return $b$

$\textsc{Kg}()$
$u \leftarrow u + 1;\ K_u \leftarrow\!\!\$\ \mathcal{K}$

$\textsc{Func}(i, \ell, X)$
return $\mathsf{F}(K_i, \ell, X)$
```

```
Game $\mathrm{PRF}^{\mathrm{rand}}_{\mathsf{F}}(\mathcal{A})$
$b \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Kg},\textsc{Func}}$
return $b$

$\textsc{Kg}()$
$u \leftarrow u + 1$

$\textsc{Func}(i, \ell, X)$
if $\mathsf{Tbl}[i, \ell, X] = \bot$
    $\mathsf{Tbl}[i, \ell, X] \leftarrow\!\!\$\ \{0,1\}^{\ell}$
return $\mathsf{Tbl}[i, \ell, X]$
```

```
Game $\mathrm{CR}_{\mathsf{F}}(\mathcal{A})$
$(\ell, (K_1, X_1), (K_2, X_2)) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{F}}$
$Y_1 \leftarrow \mathsf{F}_{K_1}(\ell, X_1)$
$Y_2 \leftarrow \mathsf{F}_{K_2}(\ell, X_2)$
if $Y_1 = \bot$ or $Y_2 = \bot$
    return false
if $(K_1, X_1) = (K_2, X_2)$
    return false
return $(Y_1 = Y_2)$
```

Figure 3.3: Games ($\mathrm{PRF}^{\mathrm{real}}, \mathrm{PRF}^{\mathrm{rand}}$) and CR for a function F.

the length of outputs match the provided output length $|\mathsf{F}(K, \ell, X)| = \ell$. To simplify notation, we sometimes use the shorthand $\mathsf{F}^{\ell}_{K}(\cdot) := \mathsf{F}(K, \ell, \cdot)$.

We define multi-user PRF security [13], extended to variable output lengths, using a pair of games $\mathrm{PRF}^{\mathrm{real}}$ and $\mathrm{PRF}^{\mathrm{rand}}$ defined in Figure 3.3, as

$$\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A}) := \Pr[\mathrm{PRF}^{\mathrm{real}}(\mathcal{A}) \Rightarrow 1] - \Pr[\mathrm{PRF}^{\mathrm{rand}}(\mathcal{A}) \Rightarrow 1]\,,$$

and define collision-resistance security, similarly extended to variable output lengths, using the game CR defined in Figure 3.3, as

$$\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A}) := \Pr[\mathrm{CR}(\mathcal{A}) \Rightarrow \mathbf{true}]\,.$$

We let $\widetilde{\mathrm{Funcs}}(\mathcal{K}, \mathcal{L}, \mathcal{X}, \mathcal{Y})$ be the set of all be the set of all families of variable-output-length functions $f : \mathcal{X} \times \mathcal{L} \to \mathcal{Y}$ satisfying $|f(\ell, X)| = \ell$ for all $\ell \in \mathcal{L}$, indexed by $K \in \mathcal{K}$.

Universal hashing. Let $\mathsf{G} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a function, defined over a key space $\mathcal{K}$, input space $\mathcal{X} \subset \{0,1\}^*$, and output space $\mathcal{Y} \subset \{0,1\}^*$, where the length of all inputs $|X|$ is upper bounded by a constant, and the length of all outputs $|Z| = \eta$ is a constant η. We will refer to such functions as hash functions and define the shorthand $\mathsf{G}_K(\cdot) := \mathsf{G}(K, \cdot)$. Then, we say that the family of hash functions $\{\mathsf{G}_K(\cdot)\}_{K \in \mathcal{K}}$ is δ-uniform if for all input

$X \in \mathcal{X}$ and all outputs $Y \in \mathcal{Y}$, it holds that

$$\Pr_{K \leftarrow\$ \mathcal{K}}[\mathsf{G}_K(X) = Y] \leq \delta\,.$$

And, we say that $\{\mathsf{G}_K(\cdot)\}_{K\in\mathcal{K}}$ is ϵ-AXU if for all all distinct inputs $X_1, X_2 \in \mathcal{X}$ and all outputs $Y \in \mathcal{Y}$, it holds that

$$\Pr_{K \leftarrow\$ \mathcal{K}}[\mathsf{G}_K(X_1) \oplus \mathsf{G}_K(X_2) = Y] \leq \epsilon\,.$$

Domain separation. We prefix distinct uses of the same function, including PRFs and AXU hashes, with unique one-byte labels, typeset in monospace like `1` and `2`, for domain separation [14].

AEAD syntax. We use a special case of the general AE3 syntax [29, 17], namely that of the CAESAR call for submissions [46], dubbed AE5 in [135]. Thus we define an AEAD scheme as a tuple of algorithms $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$, defined over a key space $\mathcal{K} \subseteq \{0,1\}^*$, public nonce space $\mathcal{N}_\mathcal{P} \subseteq \{0,1\}^*$, secret nonce space $\mathcal{N}_\mathcal{S} \subseteq \{0,1\}^*$, associated data space $\mathcal{A} \subseteq \{0,1\}^*$, message space $\mathcal{M} \subseteq \{0,1\}^*$, and ciphertext space $\mathcal{C} \subseteq \{0,1\}^*$, where:

1. $\mathsf{Enc} : \mathcal{K} \times \mathcal{N}_\mathcal{P} \times \mathcal{N}_\mathcal{S} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C}$ is a deterministic algorithm that takes a 5-tuple of a key K, public nonce N_P, secret nonce N_S, associated data A, and message M, and returns a ciphertext C.
2. $\mathsf{Dec} : \mathcal{K} \times \mathcal{N}_\mathcal{P} \times \mathcal{A} \times \mathcal{C} \to (\mathcal{M} \times \mathcal{N}_\mathcal{S}) \cup \{\perp\}$ is a deterministic algorithm that takes a 4-tuple of a key K, public nonce N_P, associated data A, and ciphertext C, and returns a message M and a secret nonce N_S or an error $\perp$.

We call the non-message inputs to Enc—the key, public nonce, secret nonce, and associated data—the *encryption context* and the non-ciphertext inputs to Dec—the key, public nonce, and associated data—the *decryption context.*

We impose correctness and *tidiness* [134] requirements on AEAD which requires Enc and Dec to be inverses of each other. Namely for all $(K, N_P, N_S, A, M, C) \in \mathcal{K} \times \mathcal{N}_\mathcal{P} \times \mathcal{N}_\mathcal{S} \times \mathcal{A} \times \mathcal{M} \times \mathcal{C}$:

$$\mathsf{Enc}(K, N_P, N_S, A, M) = C \implies \mathsf{Dec}(K, N_P, A, C) = (M, N_S),$$
$$\mathsf{Dec}(K, N_P, A, C) = (M, N_S) \neq \bot \implies \mathsf{Enc}(K, N_P, N_S, A, M) = C.$$

We also require that the lengths of keys $|K| =: k$, public nonces $|N_P| =: n_p$, and secret nonces $|N_S| =: n_s$ be public constants, and the length of the ciphertext $|C| =: \mathsf{clen}(|M|)$ to be computable by a deterministic public function clen of the length of the message $|M|$. And define the *ciphertext overhead* to be the function that on input m returns the difference $\mathsf{clen}(m) - m$.

Tag-based AEAD syntax. An AEAD scheme $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$ is *tag-based* [52, 17] if it specifies a tag length τ and its ciphertexts C can be split into a ciphertext core C^* and an authentication tag T via $C^* \| T \leftarrow C$, meaning the last τ bits of the ciphertext is the tag. Additionally, we ask that there is a partial decryption algorithm ParDec in terms of which $\mathsf{Dec}(K, N_P, A, C)$ is defined by

```
$C^* \| T \leftarrow C$ ; $(M, N_S, T_d) \leftarrow \mathsf{ParDec}(K, N_P, A, C^*)$
If $T = T_d$ then return $(M, N_S)$ else return $\bot$.
```

We refer to T_d as the expected tag.

AEAD security. We define this in the multi-user setting following Bellare and Hoang [17]. We use the pair of games $\mathrm{NAE}^{\mathrm{real}}$ and $\mathrm{NAE}^{\mathrm{rand}}$ which are defined in Fig-

```
Game $\mathrm{NAE}^{\mathrm{real}}_{\mathsf{AEAD}}(\mathcal{A})$
$b \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{NAEKG},\mathrm{NAEENC},\mathrm{NAEVER}}$
return $b$

NAEKG()
$u \leftarrow u+1;\ K_u \leftarrow\!\!\$\ \mathcal{K}$

NAEENC($i, N_P, N_S, A, M$)
if $(i, N_P, N_S) \in \mathcal{Q}_N$ then return $\perp$
$C \leftarrow \mathsf{AEAD.Enc}(K_i, N_P, N_S, A, M)$
$\mathcal{Q}_D \leftarrow \mathcal{Q}_D \cup \{(i, N_P, A, C)\}$
$\mathcal{Q}_N \leftarrow \mathcal{Q}_N \cup \{(i, N_P, N_S)\}$
return $C$

NAEVER($i, N_P, A, C$)
if $(i, N_P, A, C) \in \mathcal{Q}_D$ then return $\perp$
$(M, N_S) \leftarrow \mathsf{AEAD.Dec}(K_i, N_P, A, C)$
if $(M, N_S) \neq \perp$ then return true
else return false
```

```
Game $\mathrm{NAE}^{\mathrm{rand}}_{\mathsf{AEAD}}(\mathcal{A})$
$b \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{NAEKG},\mathrm{NAEENC},\mathrm{NAEVER}}$
return $b$

NAEKG()
$u \leftarrow u+1$

NAEENC($i, N_P, N_S, A, M$)
if $(i, N_P, N_S) \in \mathcal{Q}_N$ then return $\perp$
$C \leftarrow\!\!\$\ \{0,1\}^{\mathsf{clen}(|M|)}$
$\mathcal{Q}_D \leftarrow \mathcal{Q}_D \cup \{(i, N_P, A, C)\}$
$\mathcal{Q}_N \leftarrow \mathcal{Q}_N \cup \{(i, N_P, N_S)\}$
return $C$

NAEVER($i, N_P, A, C$)
if $(i, N_P, A, C) \in \mathcal{Q}_D$ then return $\perp$
return false
```

Figure 3.4: Games ($\mathrm{NAE}^{\mathrm{real}}, \mathrm{NAE}^{\mathrm{rand}}$) for an AEAD scheme [17].

ure 3.4, and then set

$$\mathsf{Adv}^{\mathrm{nae}}_{\mathsf{AEAD}}(\mathcal{A}) := \Pr[\mathrm{NAE}^{\mathrm{real}}(\mathcal{A}) \Rightarrow 1] - \Pr[\mathrm{NAE}^{\mathrm{rand}}(\mathcal{A}) \Rightarrow 1]\,.$$

To prevent trivial victories, we require adversaries to be *valid*—it may not query NAEENC with the same input twice, and it may not query NAEVER on an output of NAEENC. The latter condition is enforced with the set $\mathcal{Q}_D$. In addition, we require adversaries to be *nonce-respecting*—for a given user i, it may not query NAEENC with the same nonce (N_P, N_S). This is enforced with the set $\mathcal{Q}_N$.

We say that an adversary is *orderly* [44, 22, 15, 17] if its NAEVER queries are made after all the NAEENC queries, all the NAEVER queries are made at once (i.e., NAEVER queries may depend on NAEENC queries but not on other NAEVER queries), and it returns **true** if any of the NAEVER queries returns **true**. We find it useful to restrict to orderly adversaries and the following lemma of [17] upper bounds the advantage loss of such a restriction.

```
Game CMT($\mathcal{A}$)
$((K_1, N_{P1}, A_1), (K_2, N_{P2}, A_2), C) \leftarrow\!\!\$\ \mathcal{A}$
$(M_1, N_{S1}) \leftarrow \mathsf{AEAD.Dec}(K_1, N_{P1}, A_1, C)$
$(M_2, N_{S2}) \leftarrow \mathsf{AEAD.Dec}(K_2, N_{P2}, A_2, C)$
if $(M_1, N_{S1}) = \bot$ or $(M_2, N_{S2}) = \bot$ then return false
return $(K_1, N_{P1}, A_1) \neq (K_2, N_{P2}, A_2)$
```

Figure 3.5: Game CMT for an AEAD scheme [15, 52].

Lemma 8 (Lemma 3.2 in [17]). *Let* AEAD *be an AEAD scheme with constant ciphertext overhead* τ*. Let* $\mathcal{A}$ *be an* NAE *adversary making* q *queries, with at most* q_v *verification queries, to* u *users and at most* B *queries per user. Then, we can construct an orderly adversary* $\mathcal{B}$ *such that* $\mathsf{Adv}^{\mathrm{nae}}_{\mathsf{AEAD}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{nae}}_{\mathsf{AEAD}}(\mathcal{B}) + \frac{q_v}{2^\tau}$*. Adversary* $\mathcal{B}$ *makes at most* q *queries, with at most* q_v *verification queries, to* u *users and at most* B *queries per user, and has running time similar to* $\mathcal{A}$*.*

Committing security. We focus on the strongest commitment notion *context commitment* [15, 52] which requires the ciphertext to commit to the entire decryption context. In Figure 3.5 we define security. The outputs $(K_1, N_{P1}, A_1), (K_2, N_{P2}, A_2)$ of $\mathcal{A}$ are required to be in $\mathcal{K} \times \mathcal{N}_{\mathcal{P}} \times \mathcal{A}$. We define the following advantage measure

$$\mathsf{Adv}^{\mathrm{cmt}}_{\mathsf{AEAD}}(\mathcal{A}) := \Pr[\mathrm{CMT}(\mathcal{A}) \Rightarrow \mathbf{true}]\,.$$

3.3 The Xor-then-Hash Transform

In this section, we introduce a new transform for building a context committing AEAD called Xor-then-Hash (XtH). It maps a tag-based AEAD scheme AEAD and a collision-resistant hash function F to a context committing AEAD scheme $\mathsf{XtH}[\mathsf{F}, \mathsf{AEAD}]$.

Our starting point is the CTY transform of [17], which is a refinement of the CTX

transform of [52]. Consider a (non-context-committing) tag-based AEAD scheme with encryption $\mathsf{AEAD.Enc}(K, N_P, N_S, A, M)$ outputting a ciphertext-tag pair (C, T). The CTY transform works by running $(C, T) \leftarrow \mathsf{AEAD.Enc}(K, N_P, N_S, \varepsilon, M)$, where ε denotes empty associated data, and then $T^* \leftarrow H(K, N_P, N_S, A, T)$. The output is (C, T^*). This was shown to achieve NAE security in the random oracle model assuming an NAE-secure AEAD scheme and to achieve CMT security assuming H is CR.

Our primary goal is to reduce the number of bits hashed by H to generate T^*; our baseline is CTY, which hashes $|K| + |N_P| + |N_S| + |A| + |T|$ bits. Omitting any of K, (N_P, N_S), A, or T is insecure — for example, omitting (N_P, N_S) might be tempting as T already depends on the nonce, and so NAE security would still be achieved. But CMT security would not. Another straw approach would be to somehow compress $(N_P, N_S), A, T$ before hashing, for example by applying a universal hash function G to them before applying H. Since universal hashes are faster than cryptographic hash functions, this would be a net performance improvement. However, this again would not achieve CMT, as an adversary that knows the key used for G could find A, A' such that $\mathsf{G}((N_P, N_S), A, T) = \mathsf{G}((N_P, N_S), A', T)$.

First, we observe that instead of hashing the associated data A and the tag T, it suffices to hash their bitwise XOR $A \oplus T$ and still achieve CMT security. Assume for the moment that $|A| = |T|$ is fixed. Then we claim that the tag computation $T^* = H(K, N_P, N_S, A \oplus T)$ achieves CMT. At first glance, this seems incorrect, since an adversary can easily find A, A' and T, T' such that $A \oplus T = A' \oplus T'$. But actually the adversary does not have the freedom to choose the tags T or T' arbitrarily, as tags are a deterministic function of the key, nonce, and ciphertext C. This observation leads to a simple but perhaps counter-intuitive proof of CMT security assuming H is collision resistant (across all its inputs, including K). Later in this section, we will extend this

```
XtH[F, AEAD].Enc(K, N_P, N_S, A, M)
    $L \leftarrow \mathsf{F}_K^k(1)$
    $C^* \| T \leftarrow \mathsf{AEAD.Enc}(L, N_P, N_S, \varepsilon, M)$
    $T^* \leftarrow \mathsf{F}_K^{2\kappa}(2, N_P, \mathsf{ixor}(A, T))$
    return $C^* \| T^*$

XtH[F, AEAD].Dec(K, N_P, A, C^* ‖ T^*)
    $L \leftarrow \mathsf{F}_K^k(1)$
    $(M, N_S, T) \leftarrow \mathsf{AEAD.ParDec}(L, N_P, \varepsilon, C^*)$
    $T' \leftarrow \mathsf{F}_K^{2\kappa}(2, N_P, \mathsf{ixor}(A, T))$
    if $T' = T^*$ then return $(M, N_S)$
    else return $\perp$
```

Figure 3.6: **Definition of** $\mathsf{XtH}[\mathsf{F}, \mathsf{AEAD}]$**.** Here, $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{ParDec})$ is a tag-based AEAD with key length k, $\mathsf{F} : \{0,1\}^{2\kappa} \times \mathcal{L} \times \{0,1\}^{**} \to \{0,1\}^*$ is a variable-output-length collision-resistant PRF with length space $\mathcal{L} = \{k, 2\kappa\}$, and the function ixor is defined in eq. (3.1).

idea to $|A| \neq |T|$ by replacing the bitwise XOR with an unequal-length generalization ixor. As a result, we have reduced the number of bits we need to hash for CMT security to $|K| + |N_P| + |N_S| + \max\{|A| + 1, |T|\}$ bits. This means that for short associated data $|A| < |T|$, which are popular in practice, we incur no cryptographic overhead from A.

Second, we observe that if the inner tag T has length at least $\min(\kappa + n_s, 2\kappa)$ for security parameter κ, then we can omit hashing the secret nonce N_S, and rely on the inner tag T to commit to N_S. This length assumption isn't necessary for CMT security but necessary for NAE security. It corresponds to a birthday attack on the secret nonces to find an inner tag collision and thereby distinguish the outer tag from a random one. We describe this attack in more detail in Appendix 3.8.2. Notice that we never need to extend the length of T by more than n_s, so we never hash more bits than individually hashing T and N_S; and if $n_s > \kappa$, we hash fewer bits. Combining with the previous observation, we have reduced the number of bits we need to hash for CMT security to $|K| + |N_P| + \max\{|A| + 1, \min(\kappa + n_s, 2\kappa)\}$ bits.

We also take this opportunity to avoid the random oracle model by using H to derive a subkey L for the base AEAD, in addition to the tag T^*.

Specification of XtH. Let κ be the security parameter. Let $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{ParDec})$ be a tag-based AEAD with key length k and fixed secret nonce and public nonce lengths, and let $\mathsf{F} : \{0,1\}^{2\kappa} \times \mathcal{L} \times \{0,1\}^{**} \to \{0,1\}^{*}$ be a variable-output-length collision-resistant PRF with length space $\mathcal{L} = \{k, 2\kappa\}$. The transform is specified in Figure 3.6.

The function $\mathsf{ixor}(A, T)$ on the associated data A and the tag T is designed to minimize output length while allowing recovery of A given both T and $\mathsf{ixor}(A, T)$. The latter property is crucial for security (see Theorem 9). If T and A were promised to be equal length bitstrings, then a simple xor would suffice. However, in practice, the length of A is not fixed, so we define the following encoding

$$\mathsf{ixor}(A, T) := (A \,\|\, 10^{\ell-|A|-1}) \oplus (T \,\|\, 0^{\ell-|T|}) \,, \tag{3.1}$$

where $\ell = \max(|A| + 1, |T|)$.

At first glance, hashing $\mathsf{ixor}(A, T)$ instead of $A \| T$ and not hashing N_S seems counterintuitive. To see why XtH provides CMT security, note that (1) T^* is a commitment of $\mathsf{ixor}(A, T)$, N_P, and K, and (2) $(K, C^* \,\|\, T^*, N_P, \mathsf{ixor}(A, T))$ is a commitment of (N_S, A, M) because we can recover the latter from the former. The following result shows that XtH achieves CMT security; we provide the proof in Section 3.8.1.

Theorem 9. *Let* AEAD *be a tag-based AEAD and let* $\mathsf{F} : \{0,1\}^{2\kappa} \times \mathcal{L} \times \{0,1\}^{**} \to \{0,1\}^{*}$ *be a variable-output-length collision-resistant PRF with length space* $\mathcal{L} = \{k, 2\kappa\}$*. Then for any CMT adversary* $\mathcal{A}$ *we can construct a CR adversary* $\mathcal{B}$ *such that*

$$\mathsf{Adv}^{\mathrm{cmt}}_{\mathsf{XtH[F,AEAD]}}(\mathcal{A}) \le \mathsf{Adv}^{\mathrm{cr}}_{\mathsf{F}}(\mathcal{B}) \,,$$

and $\mathcal{B}$ *emulates* $\mathcal{A}$ *and makes two additional calls to each of* F *and* $\mathsf{AEAD.ParDec}$*.*

NAE security of XtH. We now transition to proving the NAE security of XtH. The traditional approach would be to follow prior work and prove that XtH preserves NAE

```
Game NAX^real(A)
    $b \leftarrow\!\!\$ \mathcal{A}^{\text{Kg,Enc,Ver}}$    // $\mathcal{A}$ must be orderly
    return $b$

Procedure NaxKg()
    $u \leftarrow u + 1;\ K_u \leftarrow\!\!\$ \mathcal{K}$

Procedure NaxEnc($i, N_P, N_S, A, M, X$)
    if $(i, N_P, N_S) \in \mathcal{Q}_N$ then return $\perp$
    $C^* \parallel T \leftarrow \mathsf{AEAD.Enc}(K_i, N_P, N_S, A, M)$
    $\mathcal{Q}_D \leftarrow \mathcal{Q}_D \cup \{(i, N_P, A, C^*, X)\}$
    $\mathcal{Q}_N \leftarrow \mathcal{Q}_N \cup \{(i, N_P, N_S)\}$
    if $T = \perp$ then return $\perp$
    $\mathsf{win} \leftarrow (T \oplus X \in W_{i,N_P})$
    $W_{i,N_P} \leftarrow W_{i,N_P} \cup \{T \oplus X\}$
    $S_{i,N_P}[N_S] \leftarrow T \oplus X$
    return $C^*, \mathsf{win}$

Procedure NaxVer($i, N_P, N_S{}^*, A, C^*, X$)
    if $(i, N_P, A, C^*, X) \in \mathcal{Q}_D$ then return $\perp$
    $(M, N_S, T_d) \leftarrow \mathsf{AEAD.ParDec}(K_i, N_P, A, C^*)$
    if $(M, N_S, T_d) = \perp$ then return false
    if $S_{i,N_P}[N_S{}^*] = \perp$ then return false
    if $T_d \oplus X = S_{i,N_P}[N_S{}^*]$ then return true
    else return false
```

```
Game NAX^rand(A)
    $b \leftarrow\!\!\$ \mathcal{A}^{\text{Kg,Enc,Ver}}$    // $\mathcal{A}$ must be orderly
    return $b$

Procedure NaxKg()
    $u \leftarrow u + 1$

Procedure NaxEnc($K, N_P, N_S, A, M, X$)
    if $(i, N_P, N_S) \in \mathcal{Q}_N$ then return $\perp$
    $C^* \leftarrow\!\!\$ \{0,1\}^{|M|}$
    $\mathcal{Q}_D \leftarrow \mathcal{Q}_D \cup \{(i, N_P, A, C^*, X)\}$
    $\mathcal{Q}_N \leftarrow \mathcal{Q}_N \cup \{(i, N_P, N_S)\}$
    return $C^*$, false

Procedure NaxVer($i, N_P, N_S{}^*, A, C^*, X$)
    if $(i, N_P, A, C^*, X) \in \mathcal{Q}_D$ then return $\perp$
    return false
```

Figure 3.7: **NAX security.** Differences to NAE (Figure 3.4) are highlighted.

security, thereby requiring the underlying AEAD scheme also be NAE-secure. However, it turns out that NAE security is unnecessarily strong for XtH. This is because XtH does not reveal the raw authentication tag of the underlying AEAD. Still, XtH reveals authenticated tag collisions for different secret nonces $N_{S1} \neq N_{S2}$ for the same key K and public nonce N_P. This then begs the question of what property is sufficient for the AEAD used in our transform.

To explore this, we define a new notion that we call NAX security. For schemes without secret nonces, NAX is strictly weaker than NAE. For schemes with secret nonces, it is incomparable. We then show that the XtH transform only requires NAX security of the base AE scheme to provide NAE security.

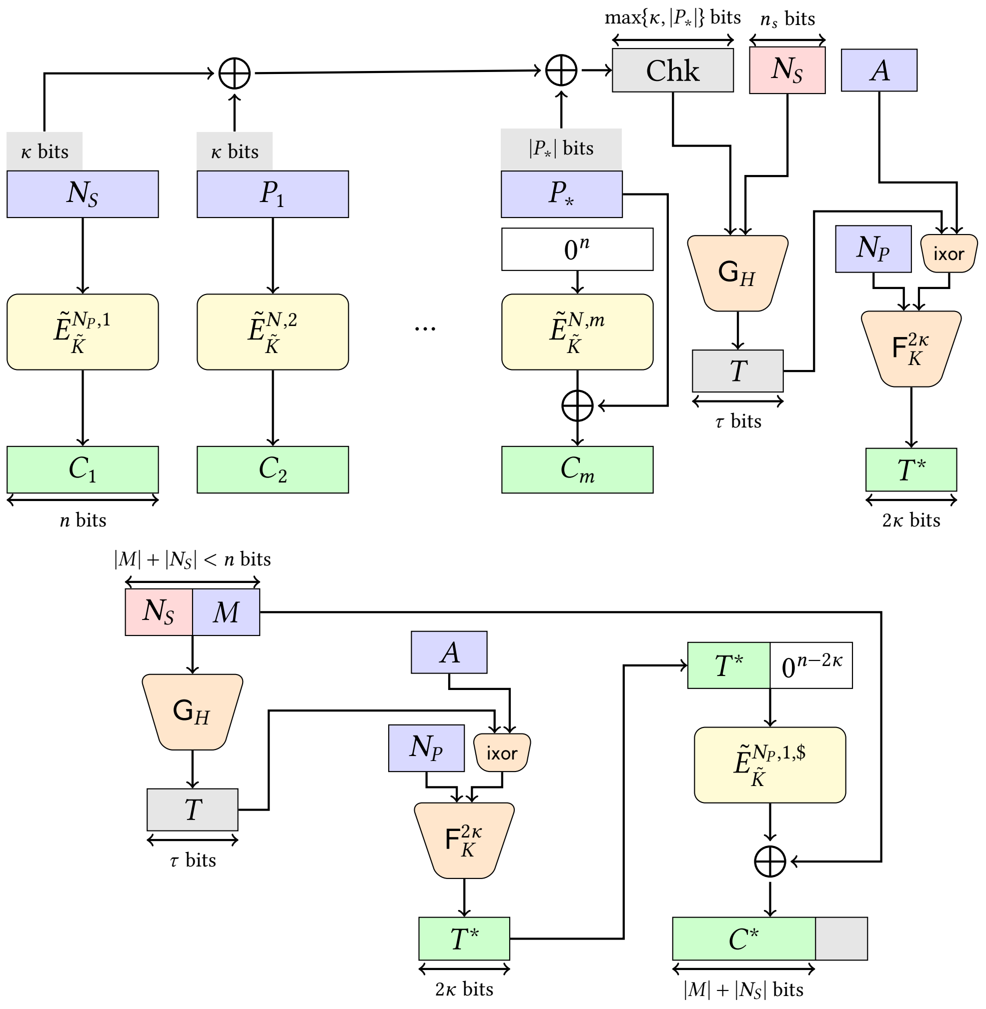

Figure 3.8: **Illustration of** $\Theta\mathsf{CH}[\tilde{E}, \mathsf{G}, \mathsf{F}]$**.** (Top) encryption using OCH-core when the secret nonce length $|N_S| = w$ and the message length $|M|$ not a multiple of n. (Bottom) encryption using OCH-tiny when $|M| + |N_S| < w$.

The NAX notion. The NAX notion formalizes the assumptions XtH makes on the underlying AEAD. First, it assumes that the authentication tag has AXU-like (almost XOR universality) security, which is strictly weaker than NAE security; the latter requires random-looking tags. Thus, as we'll see, NAX security can be achieved more efficiently. Second, it assumes that for a given user i and public nonce N_P, it is hard to find secret nonces N_{S1} and N_{S2} that result in the same inner tag T. The latter condition is because XtH does not hash the secret nonce and corresponds to the birthday attack on secret nonces described above.

NAX requires that schemes are tag-based (§3.2). We capture multi-user NAX security using a pair of games $\mathrm{NAX}^{\mathrm{real}}$ and $\mathrm{NAX}^{\mathrm{rand}}$, defined in Figure 3.7. We then define NAX advantage as

$$\mathsf{Adv}^{\mathrm{nax}}_{\mathsf{AEAD}}(\mathcal{A}) := \Pr[\mathrm{NAX}^{\mathrm{real}}(\mathcal{A}) \Rightarrow 1] - \Pr[\mathrm{NAX}^{\mathrm{rand}}(\mathcal{A}) \Rightarrow 1]\,.$$

These games are similar to those of the NAE notion (Figure 3.4) with some key differences, highlighted in Figure 3.7. In particular, in each $\textsc{NaxEnc}$ query, in addition to the usual (i, N_P, N_S, A, M), the adversary must also specify a τ-bit offset X. In the real game, it reveals the ciphertext core C^* (instead of revealing the full ciphertext $C^* \parallel T$) and a boolean $\mathsf{win} \leftarrow (T \oplus X \in W_{i,N_P})$, where W_{i,N_P} is the set of previously encountered $T \oplus X$ values. The game also saves $S_{i,N_P}[N_S] \leftarrow T \oplus X$. In the random game, the $\textsc{NaxEnc}$ oracle simply outputs a uniformly random ciphertext core C^* and $\mathsf{win} \leftarrow \mathbf{false}$. Next, for each $\textsc{NaxVer}$ query, in addition to the usual (i, N_P, A), the adversary must now specify a ciphertext core C^* (instead of the full ciphertext), a secret nonce guess ${N_S}^*$, and an offset X. In the real game, it returns **true** if and only if $\mathsf{AEAD.ParDec}$ succeeds in producing a tag T_d such that $(T_d \oplus X) = S_{i,N_P}[N_S]$. In the random game, it always returns **false**.

Like NAE, we require adversaries to be *valid*—that is, not query $\textsc{NaxEnc}$ with the same input twice or query $\textsc{NaxVer}$ on an output of $\textsc{NaxEnc}$; and to be *nonce-respecting*—that is, for a given user i, not query $\textsc{NaxEnc}$ with the same nonce (N_P, N_S) twice. Finally, unlike NAE, we additionally require NAX adversaries to be *orderly*—that is, make all the $\textsc{NaxVer}$ queries at once after all the $\textsc{NaxEnc}$ queries—this is a mild restriction since all adversaries can be turned into orderly adversaries with small loss (see Theorem 8), but nevertheless simplifies proofs and weakens NAX relative to NAE.

We emphasize that NAX does not imply NAE, due to the strictly weaker assumption on the tag, and give an explicit scheme that is NAX secure but has a two query NAE attack in Section 3.8.3.

Proposition 10 below shows that, on the other hand, NAE implies NAX. The term $q \cdot \min(B, 2^{n_s}) \cdot 2^{-\tau}$ corresponds to the birthday attack on secret nonces described above. We prove this proposition in Appendix 3.8.4.

Proposition 10. *Let* AEAD *be a tag-based AEAD with tag length* τ *and secret nonce length* n_s*. Then for any NAX adversary* $\mathcal{A}$ *making* q *queries in total and at most* B *queries per user, we can construct an NAE adversary* $\mathcal{B}$ *such that*

$$\mathsf{Adv}^{\mathrm{nax}}_{\mathsf{AEAD}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{nae}}_{\mathsf{AEAD}}(\mathcal{B}) + \frac{q \cdot \min(B, 2^{n_s})}{2^{\tau}},$$

where adversary $\mathcal{B}$ *has the same query complexity as* $\mathcal{A}$ *and similar time complexity.*

XtH promotes NAX to NAE. In Theorem 11 we show that XtH promotes a NAX secure scheme to a NAE secure scheme, and preserves NAE security. The proof is in Section 3.8.6.

Theorem 11. *Let* $\mathcal{K}_{\mathsf{XtH}} := \{0,1\}^{2\kappa}$ *and* $\mathcal{A}_{\mathsf{XtH}}$ *be the key space and associated data space for* XtH*, respectively. Let* AEAD *be a tag-based AEAD with key length* k*, tag length* τ*, public nonce length* n_p*, and secret nonce length* n_s*. Let* $\mathsf{F} : \{0,1\}^{2\kappa} \times \mathcal{L} \times \{0,1\}^{**} \to \{0,1\}^*$ *be a variable-output-length collision-resistant PRF with length space* $\mathcal{L} = \{k, 2\kappa\}$*. Let* $\mathcal{A}$ *be an NAE adversary making* q *queries, with* q_v *verification queries. Assume* $\mathcal{A}$ *queries* u *users, making at most* B *queries (encryption and verification) per user. Then, we can construct a PRF adversary* $\mathcal{B}$ *and an NAX adversary* $\mathcal{C}$ *such that*

$$\mathsf{Adv}^{\mathrm{nae}}_{\mathsf{XtH[F,AEAD]}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{B}) + \mathsf{Adv}^{\mathrm{nax}}_{\mathsf{AEAD}}(\mathcal{C}) + \frac{2qB}{2^{2\kappa}}.$$

Adversary $\mathcal{B}$ makes at most q PRF queries to u users with at most B queries per user. Adversary $\mathcal{C}$ makes at most q NAX queries, with at most q_v verification queries, to u users and at most B queries (encryption and verification) per user. Adversaries $\mathcal{B}$ and $\mathcal{C}$ have running time similar to $\mathcal{A}$.

3.4 The OCH AEAD Scheme

We specify the OCH scheme in two steps. We first specify a TBC-based scheme $\Theta\mathsf{CH}$, and then we instantiate it with a permutation-based TBC called OCT.

Underlying parameters and components. For security parameter $\kappa \in \{64, 128, 256\}$, choose a block size $w \geq 2\kappa$, TBC key length $\tilde{k} \geq 2\kappa$, AXU hash key length $h \geq 2\kappa$, public nonce length $n_p \leq w - 2$, secret nonce length $n_s \leq w$, and total nonce len $(n_p + n_s) \leq w$. Define the key space $\mathcal{K} = \{0,1\}^{2\kappa}$, public nonce space $\mathcal{N}_\mathcal{P} = \{0,1\}^{n_p}$, secret nonce space $\mathcal{N}_\mathcal{S} = \{0,1\}^{n_s}$, nonce space $\mathcal{N} = \{0,1\}^{n_p+n_s}$. Let $\mathsf{maxBlocks} \leq 2^{\kappa-4}$ be an upper bound on the number of blocks that $\Theta\mathsf{CH}$ supports, and let $r = \log_2(\mathsf{maxBlocks}) + 1$ be one more than the number of bits required to encode the block number $i \in \{1, \ldots, \mathsf{maxBlocks}\}$ (we will use the extra bit is to denote whether there is a partial block.)

$\Theta\mathsf{CH}$ uses three underlying components whose instantiations we will discuss later. First is a TBC with a structured tweak space. Let $\tilde{E} : \{0,1\}^{\tilde{k}} \times \mathcal{T} \times \{0,1\}^w \to \{0,1\}^w$ for some TBC key length $\tilde{k} \geq 2\kappa$ and with tweak space

$$\mathcal{T}_{\Theta\mathsf{CH}} := (\mathcal{N}_\mathcal{P} \times \{\varepsilon, \star, \$\}) \cup (\mathcal{N}_\mathcal{P} \times \mathcal{N}_\mathcal{S} \times \{0, 1, \ldots, \mathsf{maxBlocks}\} \times \{\varepsilon, \star, \$\}), \qquad (3.2)$$

where we can equivalently view $\{\varepsilon, \star, \$\}$ as $\{0, 1, 2\}$. When the tweak uses ε we omit it from the notation. The second component is an ϵ-AXU hash function $\mathsf{G} : \{0,1\}^h \times \{0,1\}^* \to \{0,1\}^\tau$, and the third is a variable-output-length collision-resistant PRF $\mathsf{F} : \{0,1\}^{2\kappa} \times \mathcal{L} \times \{0,1\}^{**} \to \{0,1\}^*$ with length space $\mathcal{L} = \{k, 2\kappa\}$ where $k := \tilde{k} + h$. The length space reflects that we will be generating a TBC key and an AXU hash key, as well as tags, using F.

The TBC-based scheme $\mathsf{\Theta CH}$. We take a top-down approach in our explanation of the scheme. $\mathsf{\Theta CH}$ is the composition of two encryption algorithms, one for very short messages and one for longer messages. Diagrams depicting encryption appear in Figure 3.8 and pseudocode appears in Figure 3.10.

When encrypting very short messages, e.g., messages M with length $|M| < w + n_s$, we use a construction $\mathsf{\Theta CH\text{-}tiny}$. It takes an approach inspired by SIV [149]. First, it generates a context-committing tag T^* using the AXU hash G and the CR PRF F, similar to XtH. Then it uses this tag T^* as a synthetic IV to encrypt the message M and the secret nonce N_S. On decryption, it uses the provided tag T^* to recover the message M and the secret nonce N_S, then recomputes and checks the tag. Notice that the tag T^* non-invertibly depends on the secret nonce so enables full nonce-hiding security, and since T^* is provided to decryption, we can still decrypt.

For all other situations, we use another construction $\mathsf{\Theta CH\text{-}core}$ which composes the XtH transform with a new base AEAD scheme $\mathsf{\Theta CX}$ that is tailored for the XtH transform. We explain it next. Note first, however, that the schemes $\mathsf{\Theta CH\text{-}tiny}$ and $\mathsf{\Theta CH\text{-}core}$ share the same TBC key, but use disjoint tweaks to achieve domain separation.

The $\mathsf{\Theta CX}$ construction. For all but tiny messages, $\mathsf{\Theta CH}$ utilizes an only-NAX-secure mode called $\mathsf{\Theta CX}$ to which we apply XtH. $\mathsf{\Theta CX}$ uses the same TBC and also the AXU

```
$\Theta\mathsf{CX}[\tilde{E},\mathsf{G}].\mathsf{Enc}((\tilde{K},H),N_P,N_S,M)$
$P_1,\ldots,P_m,P_* \leftarrow N_S \,\|\, M$   // $|P_i| = w$
$C_1 \leftarrow \tilde{E}_{\tilde{K}}^{N_P}(P_1)$
$\mathrm{Chk} \leftarrow P_1[1:\kappa]$
for $i \leftarrow 2..m$ do
    $C_i \leftarrow \tilde{E}_{\tilde{K}}^{N_P,N_S,i}(P_i)$
    $\mathrm{Chk} \leftarrow \mathrm{Chk} \oplus P_i[1:\kappa]$
$C \leftarrow C_1 \,\|\, \cdots \,\|\, C_m$
if $P_* \neq \varepsilon$ then   // $|P_*| < w$
    $\mathrm{Pad} \leftarrow \tilde{E}_{\tilde{K}}^{N_P,N_S,m,\star}(0^n)$
    $C_* \leftarrow P_* \oplus \mathrm{Pad}[1:|P_*|]$
    $\mathrm{Chk} \leftarrow (\mathrm{Chk} \,\|\, 0^{\max(0,|P_*|-\kappa)}) \oplus P_*$
    $C \leftarrow C \,\|\, C_*$
$\mathsf{mlen} \leftarrow \mathsf{encode}_r(m \,\|\, (P_* \neq \varepsilon))$
$T \leftarrow \mathsf{G}_H(6,N_S,\mathrm{Chk},\mathsf{mlen})$
return $C \,\|\, T$
```

```
$\Theta\mathsf{CX}[\tilde{E},\mathsf{G}].\mathsf{ParDec}((\tilde{K},H),N_P,C^*)$
$C_1,\ldots,C_m,C_* \leftarrow C^*$   // $|C_i| = w$
$P_1 \leftarrow \tilde{D}_{\tilde{K}}^{N_P,1}(C_1);\ N_S \leftarrow P_1[1:n_s]$
$\mathrm{Chk} \leftarrow P_1[1:\kappa]$
for $i \leftarrow 2..m$ do
    $P_i \leftarrow \tilde{D}_{\tilde{K}}^{N_P,N_S,i}(C_i)$
    $\mathrm{Chk} \leftarrow \mathrm{Chk} \oplus P_i[1:\kappa]$
$M \leftarrow P_1[n_s:w] \,\|\, \cdots \,\|\, P_m$
if $C_* \neq \varepsilon$ then   // $|C_*| < w$
    $\mathrm{Pad} \leftarrow \tilde{E}_{\tilde{K}}^{N_P,N_S,m,\star}(0^w)$
    $M_* \leftarrow C_* \oplus \mathrm{Pad}[1:|C_*|]$
    $\mathrm{Chk} \leftarrow (\mathrm{Chk} \,\|\, 0^{\max(0,|M_*|-\kappa)}) \oplus M_*$
    $M \leftarrow M \,\|\, M_*$
$\mathsf{mlen} \leftarrow \mathsf{encode}_r(m \,\|\, (C_* \neq \varepsilon))$
$T \leftarrow \mathsf{G}_H(6,N_S,\mathrm{Chk},\mathsf{mlen})$
return $(M,N_S,T)$
```

Figure 3.9: **Pseudocode for $\Theta\mathsf{CX}[\tilde{E},\mathsf{G}]$.**

```
$\Theta\mathsf{CH}[\tilde{E},\mathsf{G},\mathsf{F}].\mathsf{Enc}(K,N_P,N_S,A,M)$
$(\tilde{K},H) \leftarrow \mathsf{F}_K^k(3)$
if $|N_S| + |M| < w$ then
    // ΘCH-tiny
    $P \leftarrow N_S \,\|\, M$   // $|P| < n$
    $T \leftarrow \mathsf{G}_H(4,P)$
    $T^* \leftarrow \mathsf{F}_K^{2\kappa}(5,N_P,\mathsf{ixor}(A,T))$
    $C^* \leftarrow \tilde{E}_{\tilde{K}}^{N_P,N_S,1,\$}(T^* \,\|\, 0^{w-2\kappa})[1:|P|] \oplus P$
    return $C^* \,\|\, T^*$

endif
// ΘCH-core = XtH[F, ΘCX[$\tilde{E}$, G]].Enc
$C^* \,\|\, T \leftarrow \Theta\mathsf{CX}[\tilde{E},\mathsf{G}].\mathsf{Enc}((\tilde{K},H),N_P,N_S,M)$
$T^* \leftarrow \mathsf{F}_K^{2\kappa}(2,N_P,\mathsf{ixor}(A,T))$
return $C^* \,\|\, T^*$
```

```
$\Theta\mathsf{CH}[\tilde{E},\mathsf{G},\mathsf{F}].\mathsf{Dec}(K,N_P,C^* \,\|\, T^*)$
$(\tilde{K},H) \leftarrow \mathsf{F}_K^k(3)$
if $|C^*| < w$ then
    // ΘCH-tiny
    $P \leftarrow \tilde{E}_{\tilde{K}}^{N_P,N_S,1,\$}(T^* \,\|\, 0^{n-2\kappa})[1:|C^*|] \oplus C^*$
    $N_S \leftarrow P[1:n_s];\ M \leftarrow P[n_s:|P|]$
    $T \leftarrow \mathsf{G}_H(4,P)$
    $T_d^* \leftarrow \mathsf{F}_K^{2\kappa}(5,N_P,\mathsf{ixor}(A,T))$
    if $T_d^* = T^*$ then return $(M,N_S)$
    else return $\perp$
endif
// ΘCH-core = XtH[F, ΘCX[$\tilde{E}$, G]].Dec
$(M,N_S,T) \leftarrow \Theta\mathsf{CX}[\tilde{E},\mathsf{G}].\mathsf{ParDec}((\tilde{K},H),N_P,C^*)$
$T_d^* \leftarrow \mathsf{F}_K^{2\kappa}(2,N_P,\mathsf{ixor}(A,T))$
if $T_d^* = T^*$ then return $(M,N_S)$
else return $\perp$
```

Figure 3.10: **Pseudocode for $\Theta\mathsf{CH}[\tilde{E},\mathsf{G},\mathsf{F}]$.**

hash G. When using secret nonces with length $n_s > 0$, it proceeds in a careful way, applying the TBC using the public tweak $\tilde{E}_{\tilde{K}}^{N_P,1}(P_1)$ to an input P_1 that contains the secret nonce and the first $(w - n_s)$ bits of plaintext (assuming $n_s < w$). Looking ahead to security, this secret nonce handling is a bit delicate as we must ensure that the first block of ciphertext is appropriately "randomized" by both the secret and public nonces. Subsequent blocks of plaintext are then encrypted in an OCB-style mode of operation.

Tag computation deviates from prior OCB-style modes, because we will use XtH. First, we use the "half-checksum method" of Inoue and Minematsu [99, §4] to reduce the size of the checksum to κ bits for full blocks, except for a final partial block which still needs a full checksum. Then, instead of encrypting this checksum with a TBC tweaked by the nonce, the block length, and whether there was a partial block, we AXU hash the checksum and secret nonce. We omit the public nonce from the AXU hash since we don't need it for NAX (since XtH commits to it.) Notice that all the inputs to the AXU hash have constant length (in the message size.) Also, ΘCX is optimal in terms of the number of TBC calls needed to encrypt the secret nonce and the message.

Why include the secret nonce in the AXU hash if the checksum already includes it? Otherwise, an adversary could force a checksum collision by querying $(N_S, M) = (0, 1)$ and $(N_S', M') = (1, 0)$ and, thereby, produce a tag collision across distinct secret nonces $N_S \neq N_S'$, breaking NAX security. Since we must hash N_S separately, it is tempting to omit adding its first κ bits into the checksum. But that only would works if $\kappa > n_s$; we avoid having to handle different cases here by including it.

OCT**: an optimized TBC.** We now turn to the underlying TBC. While one could use any TBC that supports the tweakspace $\mathcal{T}_{\Theta\mathsf{CH}}$ and security levels, our scheme's OCB-like [151, 114] structure has most tweaks that are "increments" of prior tweaks. In addition, we would like to support precomputation given just the secret key (since in many applications keys are used across many messages) and have quick startup for the special case of nonces that are counters (since they are widespread). We therefore specify a custom permutation-based TBC called OCT.

```
$\mathsf{OH}^{\Pi^\pm}(K, T)$:
$\triangleright \{0,1\}^{2\kappa} \times \mathcal{T}_{\Theta\mathsf{CH}} \to \{0,1\}^{2\kappa}$
$L \leftarrow \mathsf{EM}^{\Pi^\pm}.\mathsf{Enc}(K, 0^w)[1 : 2\kappa]$
match $T$                // structural pattern matching
case $(N_P, j) \leftarrow T$ then
    $R_{N_P} \leftarrow \mathsf{EM}^{\Pi^\pm}.\mathsf{Enc}(K, N_P \,\|\, 0^{w-n_P-2} \,\|\, 10)[1 : 2\kappa]$
    return $(((j+2) \cdot L) \oplus R_{N_P})$
case $(N_P, N_S, i, j) \leftarrow T$ then
    $\mathsf{Top} \leftarrow N_P \,\|\, N_S[1 : n_s - 6]$
    $\mathsf{KTop} \leftarrow \mathsf{EM}^{\Pi^\pm}.\mathsf{Enc}(K, \mathsf{Top} \,\|\, 0^{w-n_P-n_s+4} \,\|\, 01)$
    $\mathsf{Bottom} \leftarrow \mathsf{str2int}(N_S[n_s - 5 : n_s])$
    $R_{N_P,N_S} \leftarrow \mathsf{SH}(\mathsf{KTop}, \mathsf{Bottom})$
    return $(((4\gamma_i + j) \cdot L) \oplus R_{N_P,N_S})$
```

```
$\mathsf{SH}(\mathsf{KTop}, \mathsf{Bottom})$:
$\triangleright \{0,1\}^w \times [1..63] \to \{0,1\}^{2\kappa}$
if $w < (2\kappa + 64)$ then
    $\mathsf{TKTop} \leftarrow \mathsf{KTop}[1 : 2\kappa]$
    $\mathsf{KTopStretch} \leftarrow \mathsf{TKTop} \| (\mathsf{TKTop} \oplus (\mathsf{TKTop} \ll c))$
elseif $w \geq (2\kappa + 64)$ then
    $\mathsf{KTopStretch} \leftarrow \mathsf{KTop}$
return $(\mathsf{KTopStretch} \ll \mathsf{Bottom})[1 : 2\kappa]$
```

Figure 3.11: **Pseudocode for** OH. Here, $\Pi : \{0,1\}^w \to \{0,1\}^w$ is a permutation of width $w \geq 2\kappa$, γ is the standard 2κ-bit Gray code, EM is the padded-key Even-Mansour blockcipher, public nonce length $n_p \leq w - 2$, secret nonce length $n_s \leq w$, and total nonce len $(n_p + n_s) \leq w$ The choice of constant c in SH is discussed in Lemma 24.

OCT is at its core a tweakable Even-Mansour cipher [56], though we use tweaks that are less than w bits as also used in [7]. Fix a security parameter $\kappa \in \{64, 128, 256\}$, and let $\Pi : \{0,1\}^w \to \{0,1\}^w$ be a permutation with width $w \geq 2\kappa$. Then $\mathsf{OCT}[\Pi] : \{0,1\}^{2\kappa} \times \mathcal{T}_{\Theta\mathsf{CH}} \times \{0,1\}^w \to \{0,1\}^w$ is defined by

$$\mathsf{OCT}^{\Pi^\pm}[\mathsf{G}].\mathsf{Enc}(K, T, X) := \Pi(\Delta_{K,T} \oplus X) \oplus \Delta_{K,T}$$

$$\mathsf{OCT}^{\Pi^\pm}[\mathsf{G}].\mathsf{Dec}(K, T, Y) := \Pi^{-1}(\Delta_{K,T} \oplus Y) \oplus \Delta_{K,T}$$

for $\Delta_{K,T} \leftarrow \mathsf{G}_K(T)$ and where $\mathsf{G} : \{0,1\}^{2\kappa} \times \mathcal{T} \to \{0,1\}^w$ is a hash function. While any hash G with AXU security would suffice, we build a custom one called OH that enables high performance. Its specified in Figure 3.11. Looking ahead to the next section, one complexity will be analyzing security of OCT given that we use a hash G that is also built from Π.

This construction borrows some techniques from OCB [114]. It uses the finite field GF($2^{2\kappa}$) with the default minimal weight irreducible polynomial. It uses the (padded-key) Even-Mansour blockcipher (defined in Figure 3.22) to derive a subkey L by encrypting the all-zero string 0^w with its key K, and taking the first 2κ bits from the result. Then it parses its tweak $T \in \mathcal{T}_{\mathsf{OCH}}$ as either $T \to (N_P, j) \in (\mathcal{N}_\mathcal{P} \times \{0, 1, 2\})$ or $T \to (N_P, N_S, i, j) \in (\mathcal{N}_\mathcal{P} \times \mathcal{N}_\mathcal{S} \times \{1, \ldots, \mathsf{maxBlocks}\} \times \{0, 1, 2\})$ and accordingly returns

$$\mathsf{OH}^{\Pi^\pm}(K, T) = \begin{cases} (((j + 2) \cdot L) \oplus R_{N_P}) & T \to (N_P, j) \\ (((4\gamma_i + j) \cdot L) \oplus R_{N_P,N_S}) & T \to (N_P, N_S, i, j) \end{cases},$$

where addition and multiplication are in GF($2^{2\kappa}$), and γ is the standard 2κ-bit Gray code. The standard 2κ-bit Gray code γ is a permutation on the set $\{0, \ldots, 2^{2\kappa}\}$ with values $\gamma_0 = 0$, $\gamma_1 = 1$, and $\gamma_i = \gamma_{i-1} + 2^{\mathsf{ntz}(i)}$ for $i \geq 1$ where $\mathsf{ntz}(i)$ is the number of trailing zeros in the binary representation of i.[1] Since the number of trailing zeros is strictly less than the the number of bits in the binary representation of i, we have $\mathsf{ntz}(i) \leq \log_2(i)$. This implies that standard Gray code values are bounded $1 \leq \gamma_i \leq 2i$ for $i \geq 1$. Combining this with our asumption that $\mathsf{maxBlocks} \leq 2^{\kappa-4}$, gives us that all these multiplliers

$$\Lambda = \{(j + 1) \,:\, j \in \{0, 1, 2\}\} \cup \{(4\gamma_i + j) \,:\, i \in \{1, \ldots, 2^{\kappa-4}\}, j \in \{0, 1, 2\}\}$$

are unique, nonzero, and strictly less than 2^κ.

The inner offset R_{N_P} is computed as the Even-Mansour encryption of the public nonce N_P (appropriately padded and domain-separated) with the TBC key K, and taking the first 2κ bits from the result.

We take a more complicated approach to compute the inner offset R_{N_P,N_S}. In practice, we expect public nonces to be random strings and secret nonces to be structured data. In most cases, including in DTLS 1.3 [142, §4.2.3] and in QUIC [160, §5.4], this structured data includes an incrementing counter (like record number or packet num-

[1] e.g., $\mathsf{ntz}(1) = \mathsf{ntz}(\texttt{0b1}) = 0$, $\mathsf{ntz}(2) = \mathsf{ntz}(\texttt{0b10}) = 1$, $\mathsf{ntz}(3) = \mathsf{ntz}(\texttt{0b11}) = 0$, $\mathsf{ntz}(4) = \mathsf{ntz}(\texttt{0b100}) = 2$.

ber.) We leverage this structure using an OCB3 [114]-inpsired approach. First, we take the concatenation of the public nonce and the secret nonce, split the bottom six-bits as Bottom, and take the rest of the bits as Top. Then we encrypts Top with the TBC key K to get KTop. Lastly, we use a custom *Stretch-then-Shift* [114] inner hash called SH (defined in Figure 3.11) to map KTop and Bottom to the inner offset R_{N_P,N_S}.

This design shines when the N_S includes a counter. Suppose N_S has an increasing counter of at least six bits. We can rearrange bits so that the six-least significant bits of the counter are the last six bits of the secret nonce N_S. Then, $2^6 = 64$ successive encryptions will have the same Top (and increasing Bottom.) Computing KTop is one of the most computationally expensive parts of evaluating OH since it requires a permutation call, and this enables us to cache these KTop values across these 64 encryptions.

Lastly, we specifically designed OCT to have fast computation of "incremental" tweaks (successive blocks within the same message.) This is enabled by the OCB3 [114]-inspired Gray code structure of OH. Suppose we precomputed two offsets $L_\star$ and $L_\$$, and a table L of $\mathsf{Lsize} \leftarrow \log_2(\mathsf{maxBlocks})$ offsets, all by doubling the subkey L. Then, given the current offset $\Delta_{(N_P,N_S,i)}$ we can compute any of its three possible increments $\Delta_{(N_P,N_S,i,\star)}$, $\Delta_{(N_P,N_S,i,\$)}$, and $\Delta_{(N_P,N_S,i+1)}$ by xor-ing the current offset with one of the these precomputed values as

$$\begin{aligned}
\Delta_{(N_P,N_S,i,\star)} &= \Delta_{(N_P,N_S,i)} \oplus L_\star \\
\Delta_{(N_P,N_S,i,\$)} &= \Delta_{(N_P,N_S,i)} \oplus L_\$ \\
\Delta_{(N_P,N_S,i+1)} &= \Delta_{(N_P,N_S,i)} \oplus L[\mathsf{ntz}(i+1)]
\end{aligned}$$

Furthermore, these precomputed values only depend on the TBC key $\tilde{K}$ (so can be cached across encryptions), and doubling in $\mathrm{GF}(2^{2\kappa})$ reduces to bit shifts and xors (so this initialization is swift).

3.5 Security Analysis of OCH

We now turn to analyzing the security of OCH. We first analyze context commitment security and then NAE security.

CMT Security of the TBC-based scheme. For non-tiny messages, $\Theta\mathsf{CH}$ is context committing assuming F is CR—this is a corollary of the commitment analysis of XtH (Theorem 9 in Section 3.3). For tiny messages, context commitment stems from the use of F to commit to the key and public nonce, which in turn fixes the secret nonce and plaintext to a single value.

Theorem 12. *Let $\tilde{E} : \{0,1\}^{\tilde{k}} \times \mathcal{T} \times \{0,1\}^w \to \{0,1\}^w$ be TBC with width $w \geq 2\kappa$ and tweak space $\mathcal{T}_{\Theta\mathsf{CH}} = (\mathcal{N}_{\mathcal{P}} \times \{\varepsilon, \star, \$\}) \cup (\mathcal{N}_{\mathcal{P}} \times \mathcal{N}_{\mathcal{S}} \times \{0, 1, \ldots, \mathsf{maxBlocks}\} \times \{\varepsilon, \star, \$\})$. Let $\mathsf{G} : \{0,1\}^h \times \{0,1\}^{**} \to \{0,1\}^\tau$ be an ϵ-AXU hash function. Let $\mathsf{F} : \{0,1\}^{2\kappa} \times \mathcal{L} \times \{0,1\}^{**} \to \{0,1\}^*$ be a variable-output-length collision-resistant PRF with length space $\mathcal{L} = \{k, 2\kappa\}$ where $k = \tilde{k} + h$. Then for any CMT adversary $\mathcal{A}$ we can construct a CR adversary $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathrm{cmt}}_{\Theta\mathsf{CH}[\tilde{E},\mathsf{G},\mathsf{F}]}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{cr}}_{\mathsf{F}}(\mathcal{B}),$$

and $\mathcal{B}$ emulates $\mathcal{A}$ and makes up to two additional calls to F, $\tilde{E}$, G, and $\Theta\mathsf{CX}[\tilde{E}, \mathsf{G}].\mathsf{ParDec}$ each.

Proof. We construct $\mathcal{B}$ such that it wins the CR game whenever $\mathcal{A}$ wins the CMT game. $\mathcal{B}$ runs $\mathcal{A}$ and then uses F, $\tilde{E}$, and $\Theta\mathsf{CX}[\tilde{E}, \mathsf{G}].\mathsf{ParDec}$ on the output to reconstruct the inputs to the underlying PRF F. For long messages, it emulates the CR adversary from Theorem 9. We define $\mathcal{B}$ formally in Figure 3.12 and analyze its success probability next. Let's assume without loss of generality that $\mathcal{A}$ either wins and produces a valid output or loses and returns $\perp$. Then, we need to prove that $\mathcal{B}$ wins whenever $\mathcal{A}$ wins

```
Adversary $\mathcal{B}^{\mathsf{F}}$
$X \leftarrow\!\!\$\, \mathcal{A}$
**if** $X = \bot$ **then return** $\bot$
$((K_1, N_{P1}, A_1), (K_2, N_{P2}, A_2), C^* \parallel T^*) \leftarrow X$
$(\tilde{K}_1, H_1) \leftarrow \mathsf{F}^k_{K_1}(3)$; $(\tilde{K}_1, H_1) \leftarrow \mathsf{F}^k_{K_1}(3)$
**if** $|C^*| \geq n$ **then**
    $(M_1, N_{S1}, T_1) \leftarrow \Theta\mathsf{CX}[\tilde{E}, \mathsf{G}].\mathsf{ParDec}((\tilde{K}_1, H_1), N_{P1}, \varepsilon, C^*)$
    $(M_2, N_{S2}, T_2) \leftarrow \Theta\mathsf{CX}[\tilde{E}, \mathsf{G}].\mathsf{ParDec}((\tilde{K}_2, H_2), N_{P2}, \varepsilon, C^*)$
    **return** $(2\kappa, (K_1, 2, N_{P1}, \mathsf{ixor}(A_1, T_1)), (K_2, 2, N_{P2}, \mathsf{ixor}(A_2, T_2)))$
**else**
    $N_{01} \leftarrow N_{P1} \parallel 0^{n_s}$; $N_{02} \leftarrow N_{P2} \parallel 0^{n_s}$
    $P_1 \leftarrow \tilde{E}^{N_{P1},\$}_{\tilde{K}_1}(T^* \parallel 0^{n-2\kappa})[1 : |C^*|] \oplus C^*$; $P_2 \leftarrow \tilde{E}^{N_{P2},\$}_{\tilde{K}_2}(T^* \parallel 0^{n-2\kappa})[1 : |C^*|] \oplus C^*$
    $N_{S1} \leftarrow P_1[1 : n_s]$; $N_{S2} \leftarrow P_2[1 : n_s]$
    $T_1 \leftarrow \mathsf{G}_{H_1}(5, P_1)$; $T_2 \leftarrow \mathsf{G}_{H_2}(5, P_2)$
    **return** $(2\kappa, (K_1, 5, N_{P1}, \mathsf{ixor}(A_1, T_1)), (K_2, 5, N_{P2}, \mathsf{ixor}(A_2, T_2)), )$
```

Figure 3.12: Definition of CR adversary $\mathcal{B}$, used in proof of Theorem 12.

and produces a valid output.

Assume towards contradiction that $\mathcal{A}$ wins and produces a valid output with ciphertext core C^*, but $\mathcal{B}$ loses. If $|C^*| \geq w$, then $\mathcal{A}$ wins the CMT game against XtH and the statement follows from Theorem 9 (since $\mathcal{B}$ emulates Figure 3.16 in this case.) Now, suppose $|C^*| < w$. Then, by definition, for $\mathcal{B}$ to lose, either the outputs don't collide or the inputs are equal. First, if the outputs don't collide

$$\mathsf{F}^{2\kappa}_{K_1}(3, N_{P1}, \mathsf{ixor}(A_1, T_1)) \neq \mathsf{F}^{2\kappa}_{K_2}(3, N_{P2}, \mathsf{ixor}(A_2, T_2))),$$

then the contexts produce different outer tags T^* implying that one of the decryptions fails. Thus, $\mathcal{A}$ could not have won the CMT game. Second, if the inputs are equal, then

$$(K_1, 3, N_{P1}, \mathsf{ixor}(A_1, T_1)) = (K_2, 3, N_{P2}, \mathsf{ixor}(A_2, T_2)),$$

which implies that both contexts have the same key $K_1 = K_2$ and public nonce $N_{P1} = N_{P2}$. This means that the only way $\mathcal{A}$ could have won the CMT game was if $A_1 \neq A_2$. But same key and public nonce implies $P_1 = P_2$ which implies that the inner tags

$T_1 = T_2$. And $T_1 = T_2$ and $\mathsf{ixor}(A_1, T_1) = \mathsf{ixor}(A_2, T_2)$ implies that $A_1 = A_2$. Thus, $\mathcal{A}$ could not have won the CMT game. $\square$

NAE Security of the TBC-based scheme. Analysis of the NAE security of ΘCH is involved. It is immediately more subtle than that for OCB3, even for just achieving confidentiality, because OCH targets nonce-hiding while OCB3 does not. In particular, OCB3 achieves a rather direct result by reducing to the underlying TBC's security—each tweak uses the nonce and since adversaries use unique nonces, then each TBC invocation is on a distinct tweak. From there, the proof can promptly conclude confidentiality.

This approach does not work for OCH because our nonce-hiding construction means that necessarily some of the tweaks used by the TBC cannot rely on the full nonce. Thus, the TBC ends up being used on the same tweak across multiple encryption queries. We then must separately, after replacing F and $\tilde{E}$ with their ideal counterparts, perform an analysis showing that security holds despite reuse of a tweak.

Our analysis takes advantage of the modularity. Due to tweak separation for $\tilde{E}$ and domain separation for F, we can first transition to the information theoretic setting in a standard way. At this point, queries are handled by either OCH-tiny and OCH-core, depending on length, and, importantly, using completely independent underlying permutations and functions. Thus, we can reduce to the information theoretic security of the two underlying schemes. We state the bound in Theorem 13, and prove it in Section 3.9.

Theorem 13. *Let* $\tilde{E} : \{0,1\}^{\tilde{k}} \times \mathcal{T} \times \{0,1\}^w \to \{0,1\}^w$ *be TBC with width* $w \geq 2\kappa$ *and tweak space* $\mathcal{T}_{\Theta\mathsf{CH}} = (\mathcal{N}_{\mathcal{P}} \times \{\varepsilon, \star, \$\}) \cup (\mathcal{N}_{\mathcal{P}} \times \mathcal{N}_{\mathcal{S}} \times \{0, 1, \ldots, \mathsf{maxBlocks}\} \times \{\varepsilon, \star, \$\})$. *Let* $\mathsf{G} : \{0,1\}^h \times \{0,1\}^{**} \to \{0,1\}^\tau$ *be hash function that is* ϵ*-AXU for inputs of length up to*

$(w + n_s + \log_2(\mathsf{maxBlocks}) + 9)$. Let $\mathsf{F} : \{0,1\}^{2\kappa} \times \mathcal{L} \times \{0,1\}^{**} \to \{0,1\}^*$ be a variable-output-length collision-resistant PRF with length space $\mathcal{L} = \{k, 2\kappa\}$ where $k = \tilde{k} + h$. Let $\mathcal{A}$ be an NAE adversary making at most $q < 2^{2\kappa-1}$ queries totalling σ blocks, with at most q_v verification queries. Assume $\mathcal{A}$ queries u users, making at most B queries (encryption and verification) per user totalling s blocks per user. Then, we can construct a PRF adversary $\mathcal{B}$ and TPRP adversary $\mathcal{C}$ such that

$$\begin{aligned}\mathsf{Adv}^{\mathrm{nae}}_{\Theta\mathsf{CH}[\tilde{E},\mathsf{G},\mathsf{F}]}(\mathcal{A}) \leq & \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{B}) + \mathsf{Adv}^{\mathrm{tprp}}_{\tilde{E}}(\mathcal{C}) + \frac{2q_v}{2^\kappa} + \frac{q(3B+1)}{2^{2\kappa}} \\ & + \frac{q \cdot \min(B, 2^{n_s})}{2^w} + 3q\min(B, 2^{n_s}) \cdot \epsilon\,.\end{aligned}$$

Adversary $\mathcal{B}$ makes at most $2q$ PRF queries and has running time similar to $\mathcal{A}$. Adversary $\mathcal{C}$ makes at most $u + \sigma$ TPRP queries and at most s TPRP queries per user, and has running time similar to $\mathcal{A}$.

Security of OCT. In the following Theorem 23, we establish the multi-user (strong) TPRP security of OCT in the ideal permutation model.

Theorem 14. *Fix a security parameter $\kappa \in \{64, 128, 256\}$, a block size $w \geq 2\kappa$, and let $\Pi \leftarrow_\$ \mathrm{Perm}(\{0,1\}^w)$ be a w-bit ideal permutation. Let $\mathcal{A}$ be a TPRP adversary making σ TPRP queries to all users, at most s TPRP queries per user, and p ideal permutation queries. Then,*

$$\mathsf{Adv}^{\mathrm{tprp}}_{\mathsf{OCT}}(\mathcal{A}) \leq \frac{2.5\sigma^2 + 2\sigma p + 1.5\sigma s + 3p}{2^{2\kappa}}\,.$$

We refer to Appendix 3.10 for the full proof, and discuss the ideas and nuances here.

First, we consider $\mathsf{TEM}[\Pi, H]$, which is the tweakable Even-Mansour cipher defined as $\Pi((X \oplus H_{\tilde{K}}(T)) \,\|\, 0^*) \oplus H_{\tilde{K}}(T)$ and where the hash function H (unlike OH) is independent of Π. That means that in the ideal permutation model, H does not make calls to the Π, Π^{-1} oracles. CLS [56] showed single-user strong PRP security in this setting assuming H is ϵ-AXU and δ-uniform. Theorem 23 in the Appendix gives a good bound for the multi-user setting under the same conditions.

Now, we want to conclude that $\mathsf{OCT}[\Pi]$ is secure. The difficulty is that the hash used $H = \mathsf{OH}[\Pi]$ is *not* independent of Π: it does make Π queries. We address this by using the multi-user strong SPRP security of padded-key Even-Mansour from ADMV [7] (Theorem 22) to move to a game where OH is replaced by a new procedure $\textsc{Off}$ that does not make Π queries. (Actually we only need PRP security, not strong). With this, we have transformed OCT into a $\mathsf{TEM}[\Pi, \textsc{Off}]$ to which we can apply the aforementioned Theorem 23.

Security of OCH. The following theorem bounds the multi-user NAE security of $\mathsf{OCH}[\Pi, \mathsf{G}, \mathsf{F}] = \Theta\mathsf{CH}[\mathsf{OCT}[\Pi], \mathsf{G}, \mathsf{F}]$ in the ideal permutation model, it combines Theorems 13 and 14.

Theorem 15. *Fix a security parameter $\kappa \in \{64, 128, 256\}$, a block size $w \geq 2\kappa$, an AXU hash key length h, public nonce length $n_p \leq w-2$, secret nonce length $n_s \leq w$, total nonce len $(n_p + n_s) \leq w$, maximum number of blocks $\mathsf{maxBlocks} \leq 2^{\kappa-4}$. Let $\Pi \leftarrow\$ \mathrm{Perm}(\{0,1\}^w)$ be a w-bit ideal permutation. Let $\mathsf{G} : \{0,1\}^h \times \{0,1\}^{**} \to \{0,1\}^\tau$ be a hash function that is ϵ-AXU for inputs of length up to $(w+n_s+\log_2(\mathsf{maxBlocks})+9)$. Let $\mathsf{F} : \{0,1\}^{2\kappa} \times \mathcal{L} \times \{0,1\}^{**} \to \{0,1\}^*$ be a variable-output-length collision-resistant PRF with length space $\mathcal{L} = \{k, 2\kappa\}$ where $k = 2\kappa + h$. Let $\mathcal{A}$ be an NAE adversary making at most $q < 2^{2\kappa-1}$ queries totalling σ blocks, with at most q_v verification queries. Assume $\mathcal{A}$ queries u users, making at most B queries (encryption and verification) per user totalling s blocks per user. Assume $\mathcal{A}$ makes p ideal permutation queries. Then, we can construct a PRF adversary $\mathcal{B}$ and TPRP* *adversary $\mathcal{C}$ such that*

$$\begin{aligned}\mathsf{Adv}^{\mathrm{nae}}_{\Theta\mathsf{CH}[\tilde{E},\mathsf{G},\mathsf{F}]}(\mathcal{A}) \leq & \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{B}) + \frac{2q_v}{2^\kappa} + \frac{2.5\sigma^2 + 1.5s\sigma + 2p\sigma}{2^{2\kappa}} + \frac{q(3B+1)}{2^{2\kappa}} \\ & + 3q\min(B, 2^{n_s})\cdot\epsilon + \frac{q\cdot\min(B,2^{n_s})}{2^w} + \frac{6.5\sigma u + 5pu + 2.5u^2}{2^{2\kappa}}\,.\end{aligned}$$

Adversary $\mathcal{B}$ makes at most $2q$ PRF queries, p ideal permutation queries, and has running time similar to $\mathcal{A}$.

3.6 Performance

We evaluate the performance of both XtH as a general transform and OCH as a full construction. We start by describing our concrete instantiation and our evaluation methodology, then answer the following questions:

1. How does XtH compare to prior hash-based transforms?
2. What is the performance difference between public and hidden nonces?
3. Is OCH faster than other high-security schemes that also achieve 128-bit context commitment and AE security?
4. Is OCH competitive with widely deployed, but low security schemes like AES-GCM and ChaCha20/Poly1305?

Our experimental analysis targets devices with the widely available AES instructions, accounting for 98% of the devices that participated in the March 2025 Steam survey [57] and encompassing even lighter weight platforms such as the Raspberry Pi 5 [121].

OCH instantiations. In our experiments, we fix key length, total nonce length, and tag length to be $k = \ell = \tau = 256$. We implemented and experimented with a variety of OCH instantiation choices. This included various permutation choices for the TBC: Sparkle256 [70], Areion256 [102], Areion512 [102], and the reduced round version of the 1024-bit-wide blake2b permutation [153, 10, 85].

We also explored different choices for the CR-PRF including SHA256 and SHA3-256, a Sponge [39] hash using Sparkle512, and a Sponge hash using Areion512. For the Sponge instances, we used rate $r = 256$ and capacity $c = 256$. One could use an overwrite Sponge [37] to reduce memory state by 256 bits; this won't otherwise materially affect performance. We use padding to ensure that computing $(\tilde{K}, H) \leftarrow \mathsf{F}_K^k(3)$ takes only two permutation calls, and then Sponge computation continues from the intermediate state to finalize tag computation.

For AXU hash, we use two POLYVAL (the version in [91]) instances (making $|H| = 256$) in parallel to provide ensure 128-bit security for internal tags. A discussion of this construction and its security appears in Appendix 3.12. This eschews a performance optimization possible when using only public nonces (where we could use a single POLYVAL instance); the performance benefit of customizing the hash to the instantiation is negligible given that we are only applying the AXU hash to a few blocks of data.

Experiments showed that for our target platforms, a choice of Areion256 for the TBC and Areion512 for the CR-PRF is highest performing. We call this instantiation AreionOCH, and below just refer to it as OCH. We let AreionOCH-P be OCH using 256-bit public nonces and no secret nonces. Similarly, AreionOCH-S denotes using no public nonce and a 256-bit secret nonce.

Implementing OCH. We report on our optimized implementation of OCH, which consists of about 1,000 lines of C. We also provide an unoptimized, simpler to understand reference implementation that is a bit more than 170 lines of C. Our implementation uses a modified version of the reference code for Areion [102] to achieve higher throughput by interleaving four concurrent permutation computations in order to saturate the processor's CPU pipelining. We implemented in-place buffer management, using careful pointer arithmetic to handle secret nonces which must be placed before the ciphertext.

Experimental setup. We use three platforms to represent typical hardware: an Intel NUC with an i7-1360P, a Raspberry Pi 5 with an ARM Cortex-A76, and an Apple Macbook Pro with an M2 Pro. The Intel NUC is representative of Intel servers and laptops, the Raspberry Pi is representative of phones and medium-to-high-end IoT devices, and the Macbook Pro is representative of Arm servers and laptops. All systems had greater than 1 GB of memory and `aes`, `clmul`, and `sha2` instructions.[2] In addition, the Intel platform has `vaes` and `vclmul` instructions, and the Apple platform has `sha3` instructions. Our implementations are optimized to use the fastest available instructions on each platform.

To run the benchmarks, we used BoringSSL's `speed` tool [43]. It runs the function we want to benchmark in a loop for an estimated 100 milliseconds and returns the number of successful invocations and the actual time taken in microseconds. We extended this tool to also measure CPU cycles on the supported Intel NUC and the Raspberry Pi following Pornin [139]. From these measurements, we can compute throughput in millions of bytes-per-second as $(\mathsf{msgLen} \cdot \mathsf{numCalls})/\mathsf{micros}$, and cycles-per-byte as $\mathsf{cycles}/(\mathsf{msgLen} \cdot \mathsf{numCalls})$, where msgLen is in bytes. We report on the median of four executions of the benchmark.

[2] `clmul` corresponds to `pclmulqdq` on Intel and `pmull` on Arm, and `vclmul` corresponds to `vpclmulqdq` on Intel.

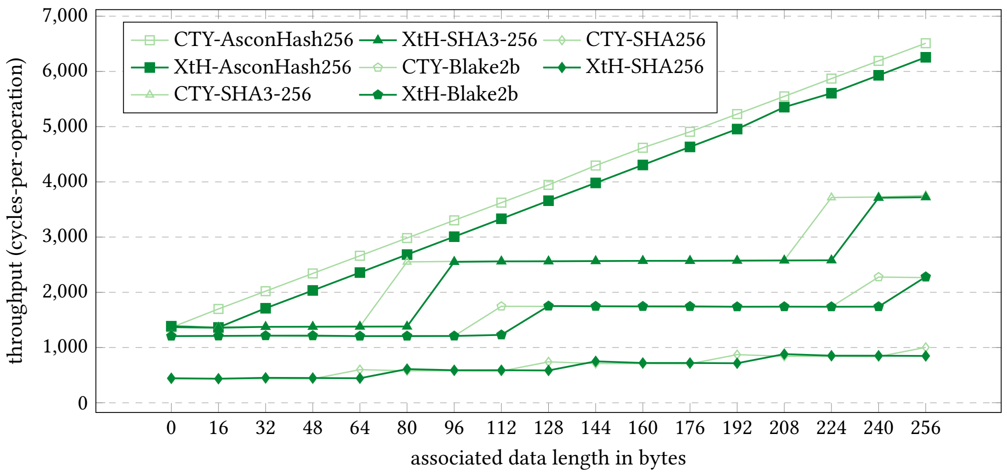

Figure 3.13: **Committing transform performance on desktop x86-64.** Throughput in CPU cycles-per-operation (y-axis, lower is faster) for encrypting a 16-byte message with varying amounts of associated data (x-axis, in bytes) on an Intel Raptor Lake processor. Solid lines indicate schemes that achieve 128-bit NAE and 128-bit CMT security, and dashed lines indicate schemes that do not.

Performance of the XtH transform Figure 3.13 compares the performance of XtH (Section 3.3) to the best prior transform in the literature CTY [52, 17] for different choices of the CR-PRF. Since these transforms only differ on the amount of data hashed, we fix the base AEAD to be AES-GCM and keep the message size constant at 16 bytes, and only vary the CR-PRF and the amount of associated data. We report on four CR-PRFs built from SHA256, Blake2b, Ascon, and SHA3-256.

Figure 3.13 (lower is better) shows how XtH outperforms CTY for some associated data lengths by up to 1.8x (when using SHA3-256). In most cases, however, the performance difference is negligible: XtH saves on performance in situations where it avoids an extra underlying primitive call (compression function or permutation) that cannot be avoided by CTY. Ultimately we conclude that XtH is, as expected, the preferred transform for rendering a scheme context committing.

Public versus secret nonces. We next turn to evaluating the performance impact of nonce hiding. OCH was engineered to make nonce hiding cheap: when one uses a secret nonce instead of a public nonce you add an additional TBC call but avoid an extra block of data processing by the CR-PRF. The difference will be small, except perhaps for small messages.

Figure 3.14 reports performance of AreionOCH-P (solid green circles) and AreionOCH-S (unfilled green circles) across the three platforms in terms of throughput in millions of bytes per second for encryption of increasingly large messages with a fixed associated data length of 13 bytes (the amount used in TLS 1.2). We focus on throughput since it demonstrates the performance trends; we provide the more standard cycles-per-byte graphs in Section 3.13.

Interestingly, we see across all three platforms that performance either does not much change depending on nonce type, or that OCH with public nonces performs slightly better. The latter effect is most visible for moderate sized messages on the two ARM systems (middle and bottom graphs). Further experimentation showed that this is due to buffer management effects, as in-place encryption for secret nonces requires writing into the next block. Nevertheless, the performance gap is minor, and using a separate buffer for the portion of the ciphertext containing the nonce might obviate it.

We also experimented with split public and secret nonces (each 128 bits). This performs slightly worse than fully secret nonces (and so public nonces), because we need to initialize OCT twice (once with the public nonce for encryption of the secret nonce, once for the remaining blocks of encryption).

OCH versus high-security schemes. For most message lengths, OCH matches or is faster than other high-security schemes. We now turn to comparing the performance of OCH with other schemes achieving similar levels of security, most relevantly 128-

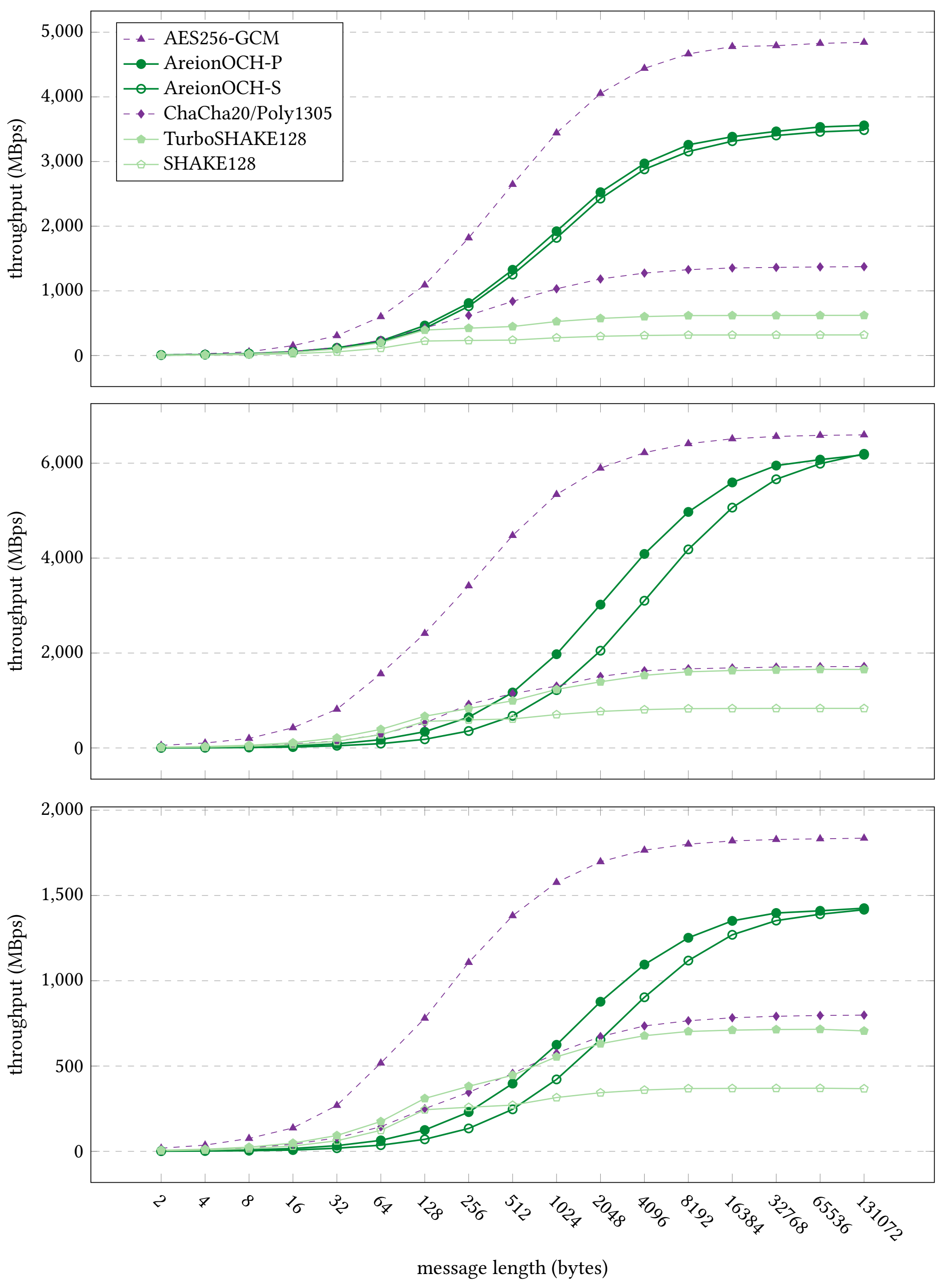

Figure 3.14: **AEAD performance.** Throughput in millions of bytes per second (y-axis, higher is better) for encrypting messages of various sizes (x-axis, in bytes) for 13 bytes of associated data on **(top)** Intel Raptor Lake, **(middle)** an Apple M2 Pro, and **(bottom)** ARM Cortex-A76. Solid lines indicate schemes that achieve 128-bit NAE and 128-bit CMT security, and dashed lines indicate schemes that do not.

	AreionOCH-S	AreionOCH-P	SHAKE128	TurboSHAKE128	AES128-GCM	AES256-GCM	ChaCha20/Poly1305	Ascon-AEAD128
		high-security			low-security			
Intel Raptor Lake	3458.8 MB/s	0.98×	**10.88**×	**5.58**×	0.62×	0.72×	**2.52**×	**11.40**×
Apple M2 Pro	5988.4 MB/s	0.99×	**7.18**×	**3.62**×	0.79×	0.91×	**3.50**×	**12.02**×
ARM Cortex-A76	1389.9 MB/s	0.99×	**3.76**×	**1.94**×	0.63×	0.76×	**1.75**×	**3.40**×

Figure 3.15: Throughput (millions of bytes per second) speedup of AreionOCH-S with respect to other AEAD schemes for 65,537 byte messages with 13 bytes of associated data and on Intel Raptor Lake, an Apple M2 Pro, and ARM Cortex-A76. AreionOCH-S is significantly faster than most schemes (bold entries with speedup >1), and competitive elsewhere.

bit NAE and context commitment security. This rules out some schemes that achieve these security goals, but at lower security levels, e.g., CTX applied to AES-GCM doesn't achieve 128-bit NAE and Ascon [67] only enjoys 64 bits of context-commitment security. We discuss such comparisons below.

We are aware of only one family of schemes, the recently proposed context committing AEAD schemes from Daemen et al. [60] built from the hash functions SHAKE128 and TurboSHAKE128. Due to the current lack of availability of source code for these AEAD schemes, we instead measure timings of the underlying hash functions (SHAKE128, TurboSHAKE128) on the same number of bytes as would be hashed in the AEAD schemes. This provides a conservative comparison (the AEAD modes add overhead beyond hashing); we confirmed this fact via communication with the authors of [60].

As can be seen, both versions of OCH perform significantly better for moderate and longer messages. For very long messages, Figure 3.15 shows OCH acheving up to 10× the throughput of SHAKE128 and 5× of TurboSHAKE128. Even on Apple M2

Pro with sha3 instruction, OCH reaches 7× the throughput of SHAKE128 and 3.5× of TurboSHAKE128. For tiny messages hashing is slightly faster, though recall that this comparison ignores other overheads for implementing AEAD using the hash functions. The crossover point is 256 bytes for AreionOCH-P and 512 bytes for AreionOCH-S.

OCH versus low-security schemes. We now turn to an *unfair* comparison of OCH to schemes that only achieve much weaker security levels. Even here OCH can outperform many schemes for a range of message lengths, and comes close to matching the performance of the fastest low-security AEAD schemes. To establish this, we performed extensive experiments with other AEAD schemes, including AES128-GCM and AES256-GCM [73]; ChaCha20/Poly1305 [136] and XChaCha20/Poly1305 [9]; Blake2b-OPP-MEM [85]; Aegis [65]; Ascon [67]; and OCB3 using AES128 [114]. We published the full gamut of performance results to `https://cryptography.run`, and only discuss representative comparisons here for brevity.

As seen in Figure 3.14, OCH outperforms ChaCha20/Poly1305 for medium and long messsages on all three platforms. For longer messages of length $2^{16} + 1$ bytes, OCH achieves 1.75× to 3.5× the throughput of ChaCha20/Poly1305. For the ARM architectures, ChaCha20/Poly1305 achieves slightly higher throughput for shorter messages, with crossover point around 1,024 bytes for AreionOCH-P and around 2,048 bytes for AreionOCH-S.

AES256-GCM is very fast on these platforms given the extensive hardware support tailored to it, and it outperforms OCH. We again emphasize that AES256-GCM is insufficiently secure for important applications as explained earlier in the paper. For longer messages, the overhead of OCH over AES-GCM can be as low as 10% (on Apple M2, against AES256-GCM), to as high as 61% (on Intel Raptor Lake, against AES128-GCM.) This shows that moving to OCH with its significantly better security does not cost much.

Figure 3.15 contains more speed-up comparisons with other AEAD schemes.

3.7 Further Related Work

Authenticated encryption (AE) unified both confidentiality and authenticity (also called integrity) in symmetric encryption and was first formally treated by [21]. Subsequently, AEAD [147, 144] brought into scope the widely understood need by practitioners to authenticate plaintext headers associated with encrypted data, the so called associated data. We group our discussion of related work in terms of design paradigms, and provide a high-level comparison of OCH with notable examples from these design paradigms in Figure 3.1.

Generic composition. Early approaches to building AEAD required protocol designers and implementers to combine separate primitives, symmetric encryption and message authentication schemes. The results were often insecure. Bellare and Namprempre give a formal analysis of generic composition, in the context of randomized AE (where the encryption is randomized) [21, 20]. Namprempre, Rogaway, and Shrimpton [134] revisited and expanded this treatment by considering different formalizations such as nonce-based AEAD. Schemes used in practice are, however, rarely generic compositions, since strictly speaking this requires separate keys for each primitive and one can achieve better efficiency by directly building AEAD from some lower level primitive, such as a block cipher.

CAU paradigm. In terms of direct construction of UNAE secure schemes, we break down the prevalent approaches seen in practice into a set of roughly four distinct design paradigms. The first is what we'll refer to as the computational almost universal or CAU paradigm, first introduced by the Galois Counter Mode (GCM) scheme of McGrew and Viega [124] and later adopted in Langley and Nir's [136] combination of ChaCha20 [34] and Poly1305 [35] into an AEAD scheme.

In this encrypt-then-MAC style design approach, one combines a stream cipher (perhaps itself built from a permutation or block cipher in a counter mode of operation) to encrypt the message. The uniqueness of a nonce guarantees a fresh stream for each encryption. Then one uses a Carter-Wegman [165, 47] style universal hash based MAC. These tend to rely on multiplications in an appropriate finite field (e.g., GF(2^{128}) with some fixed representation). These have many advantages, including high speed, mature security analyses, and the ability to precompute the pad used for encryption. But they fail to provide commitment security due to the ease of finding universal hash collisions when one knows the key. Moreover, many schemes have too-short nonces, for example AES-GCM supports only up to 96-bit nonces, barring use of random nonces. Many of these designs rely on AES, whose limited block size of 128 bits also places restrictions on achievable security. They are also not nonce-hiding.

Hash-based paradigm. A second design approach is to directly build AEAD from a single application of a cryptographic hash function [71, 68, 37]. These were first investigated under the term Duplex mode for sponge-based hash functions [39, 37] that are built from a permutation. Duplex uses internal chaining values as encryption pads. The final hash output is the tag. Since tags are therefore a collision-resistant (CR) hash of all the encryption inputs, these schemes can, in principle, achieve context committing security. But many current hash-based designs, such as Ascon [67], are tailored for performance on low-end devices and have 128-bit tags that provide at most 64 bits of commitment security. Moreover, these schemes are fundamentally sequential and do not support parallelization, and they are not nonce-hiding.

Related to hash-based paradigms is the target of compactly committing AEAD, introduced by Grubbs, Lu, and Ristenpart [86] and achieved via a hash-based construction by Dodis et al. [71]. For such schemes, just the tag itself serves as a commitment to the full decryption context and message. OCH does not strive to be compactly committing, which requires (essentially) a CR hash of the context plus the entire message [86] and therefore prevents parallelizable encryption/decryption.

We note that later work observed that for Duplex-style AEAD constructions, one can actually perform what is called full-state absorption, meaning that associated data processing can be performed (absorbed in Sponge terminology) in full n-bit blocks, i.e., XORing an entire n bit block into the full state of the Duplex [81, 59, 131]. This works for secret key settings, but tag computation is no longer a collision resistant hash of its inputs, making these approaches unsuitable for context commitment.

AEGIS is a family of AES-based AEAD schemes [167]. It can be viewed roughly as a single-pass AEAD in a Duplex-style mode, though not using a permutation but rather a function specially built using the AES round function. AEGIS offers high performance, particularly with AES hardware support, but it is not nonce-hiding, and recent attacks show how it is not key committing [101].

Permutation-based paradigms. The permutation-based hash AEAD designs above overlap with a broader set of approaches that build AEAD from (just a) public permutation. This viewpoint goes back at least to Even-Mansour [76] who suggested the seminal approach of building a block cipher via $K \oplus \Pi(K \oplus M)$ from which one can apply block cipher modes.

Instead of building a block cipher, one can use permutations to directly build noninvertible cryptographic objects, such as the variable-input-length, variable-output-length PRFs sometimes referred to as deck functions [38]. From there, one can build UNAE schemes using just black-box calls to the PRF. A number of permutation-based proposals have been built (c.f., [6, 62, 61, 98]), but these are not context committing nor do they provide nonce-hiding (with the exception of [98] which does target nonce-hiding). A very recent proposal to build context committing AEAD from decks was given in [58], and while we have not yet evaluated this concurrent work it's notable that it does not support full parallelization.

TBC-based paradigms. A fourth design approach uses tweakable blockciphers (TBCs), which originated in the work of Jutla with his IAPM construction of AE [106]. IAPM can be viewed as taking ECB mode and adding appropriately chosen XOR-based masks before and after application of the block cipher. Tags use the simple XOR checksum of plaintext blocks and then another (masked) application of the block cipher; this can be conceptually (but imprecisely) viewed as applying a PMAC-style [40] MAC to the ciphertext during decryption. The XECB construction [82] exhibited some similar design ideas around the same time. IAPM and XECB had various limitations, including that they could not encrypt messages of length not a multiple of n, which Rogaway and coauthors rectified in their offset codebook mode (OCB) construction of AE [151] that later got extended to AEAD via a judicious use of PMAC-style processing of the associated data. Ultimately these designs allow one to perform just a single cryptographic pass over data in a way that is fully parallelizable.

As such, schemes emanating from this design paradigm represent some of the fastest constructions both in theory (minimizing the number of underlying primitive calls and multiplications) and in practice on many architectures [114]. Several lightweight variants of OCB3 have been suggested as well (e.g., [155, 48, 104]). However, OCB's fast checksum-based tags are vulnerable to key committing attacks [2] and most OCB schemes use a PMAC-style PRF to process associated data, rendering them not context committing [125]. Finally, current proposals have small nonces and are not nonce hiding.

Note that the original modes (IAPM, OCB) predate and motivated the formalization of TBCs [120, 116]. But the modes are easier to understand and analyze using this intermediate abstraction. It also motivated a large number of works on building TBCs, including ones from public permutations.

Tweakable Even-Mansour. Jutla's original paper on IAPM [106] suggested that it is safe to replace the block cipher in his AEAD with a public permutation, implicitly using what would later be called tweakable Even-Mansour (this terminology was first used by [56]). Kurosawa [116] made this viewpoint more explicit suggesting a construction of the form $H_K(T) \oplus \Pi(H_K(T) \oplus M)$ for an XOR-universal hash family H [110]. Kurosawa later proved its security [115]. Cogliati, Lampe, and Seurin [56] suggested the same XOR-universal hash based construction and gave an alternative proof. OCT is similar to the tweakable EM TBC by Sasaki et al. [156], which was also used in the OCB-style minalpher lightweight AEAD. But their results don't suffice for us: their TBC uses full n-bit masks, a different mask structure, doesn't support large nonces, and their proof does not consider multi-user security.

In another line of work, researchers investigated the security of (the non-tweakable) Even-Mansour cipher using keys of length less than n bits. This viewpoint seems to have emanated out of analysis of secret-keyed Sponge-type constructions, such as Duplex, where Chang et al. [53] showed how to interpret Sponges applied to a key concatenated with a message as an iteration of Even-Mansour with shorter-than-n-bit keys. This clever observation allows various analyses of keyed Sponges [81, 7, 59, 131, 129]. But these analyses do not consider tweaks, which weren't needed in their context.

OCT can be seen as a tailored variant of Mennink's XPX paradigm for constructing a permutation-based TBC [130], which in-turn generalizes Rogaway's XEX [146] and the later Tweakable Even-Mansour (TEM) [155, 56, 85] constructions. However, while similarly motivated and designed OCT is not formally a XPX TBC since it uses padded-key Even-Mansour [7] instead of standard Even-Mansour and its hash function doesn't have the form $t_1K \oplus t_2\Pi(K)$ (where K is the TBC key and (t_1, t_2) are the tweak) that XPX expects [130, §3].

The MEM construction of Granger et al. [85] can serve as a drop-in replacement for OCT. At a high-level, OCT follows the Gray code paradigm of OCB1 [151] and OCB3 [114], while MEM follows the LFSR paradigm of Chakraborty and Sarkar [49].

Commitment and nonce-hiding. Context commitment was formalized as CMT-4 in [15] and $\mathrm{CAE}_{\mathrm{XX}}$ in [52]. It was called context committing security in [125]. These definitions were for conventional AEAD, where the decryption algorithm gets the nonce. Our work gives the first formalization for AE2 [22], where the decryption algorithm does not get the nonce.

A line of work has investigated general transforms for rendering AEAD schemes key or even context committing. A folklore approach suggested for the special case of CAU schemes is to prepend 2κ zeros to a message and verify these zeros during decryption. This provides key commitment (under suitable assumptions about the underlying block cipher / stream cipher), but not context commitment [125]. Albertini et al. [2] suggest adding key identifiers, in this case a CR hash of the key, providing key commitment. Such key identifiers were formalized as such and studied by Len et al. [117]. But again these do not provide context commitment.

Bellare and Hoang [15] give blockcipher-based transforms that add both key and context committing security to a given AEAD scheme. They also give tiny modifications of AES-GCM that achieve key and context committing security. The latter are fast, but offer only 64 bits of committing security. As noted earlier, since committing security is subject to offline attack, 128 or more bits of security, as OCH can provide, is desirable.

Chan and Rogaway [52] suggested the CTX transform where one hashes the key, nonce, associated data, and the authentication tag output by most (non-committing) AEAD schemes. Their security analysis models the hash as a random oracle, and relies on the tag being cryptographically strong. Our approach similarly hashes the key, nonce, and associated data, but using a hash that obtains efficiency improvements by omitting use of a cryptographically strong "inner" tag and instead processing a simple XOR checksum. Authenticity is guaranteed by co-designing the encryption portion with the CR-PRF, and such checksum-based tags would not be secure for other approaches to encryption (e.g., stream cipher based modes).

Similarly to transforms for rendering non-committing AEAD committing, Bellare, Ng, and Tackmann [22] proposed a number of generic ways to convert an AEAD scheme into a nonce-hiding AEAD scheme. OCH's approach is distinct from all their generic approaches, and using their transforms instead would be less efficient.

Misuse-resistance and other forms of robustness. We note that all of our discussion above has focused on UNAE schemes, i.e., ones that target confidentiality and integrity when nonces are never reused. Misuse-resistant authenticated encryption (MRAE), first suggested by Rogaway and Shrimpton [149], instead ensures that should a nonce repeat, then the AEAD falls back to just revealing plaintext and associated data equality. This goal prevents online encryption where blocks of plaintext are processed as in a streaming manner, since any MRAE scheme must process all bits of the message before the first bit of ciphertext is output.

Robust AE as defined by Hoang, Krovetz, and Rogaway [95] asks for AE for which one can specify a parameter λ, the ciphertext stretch, and the scheme should give an AEAD with that stretch that is optimally secure. Such schemes generally work in the encode-then-encipher paradigm [24], and like MRAE cannot support online encryption.

AES-based permutations. A line of work has explored building cryptographic permutations using AES instructions, due to their widespread availability on modern hardware. Simpira [92] and Haraka [109] were initial, concurrent proposals for permutations using AES instructions, with Haraka having a specific view towards hash-based signatures. In short order, Simpira v1 and Haraka v1 were broken [152, 66, 103] then updated to Simpira v2 and Haraka v2 respectively which fix the weaknesses. A recent attack significantly reduced Haraka v2's security [12]. Areion [102] is a very recent proposal which improves on the Feistel-type design of Simpira and aims to be faster than Simpira v2 by making better use of instruction pipelines.

```
Adversary $\mathcal{B}^{\mathsf{F}}$
$X \leftarrow\!\!\$\ \mathcal{A}$
if $X = \bot$ then return $\bot$
$((K_1, N_{P1}, A_1), (K_2, N_{P2}, A_2), C^* \parallel T^*) \leftarrow X$
$L_1 \leftarrow \mathsf{F}^k_{K_1}(1);\ L_2 \leftarrow \mathsf{F}^k_{K_2}(1)$
$(M_1, N_{S1}, T_1) \leftarrow \mathsf{AEAD.ParDec}(L_1, N_{P1}, \varepsilon, C^*)$
$(M_2, N_{S2}, T_2) \leftarrow \mathsf{AEAD.ParDec}(L_2, N_{P2}, \varepsilon, C^*)$
return $(2\kappa, (K_1, 2, N_{P1}, \mathsf{ixor}(A_1, T_1)), (K_2, 2, N_{P2}, \mathsf{ixor}(A_2, T_2)))$
```

Figure 3.16: Definition of CR adversary $\mathcal{B}$, used in proof of Theorem 9.

3.8 XtH Security Proofs

3.8.1 Proof of Theorem 9

We construct $\mathcal{B}$ such that it wins the CR game whenever $\mathcal{A}$ wins the CMT game. $\mathcal{B}$ runs $\mathcal{A}$ and then uses F and $\mathsf{AEAD.ParDec}$ on the output to reconstruct the inputs to the PRF F underlying XtH. We define $\mathcal{B}$ formally in Figure 3.16 and analyze its success probability next. Let's assume without loss of generality that $\mathcal{A}$ either wins and produces a valid output or loses and returns $\bot$. Then, we need to prove that $\mathcal{B}$ wins whenever $\mathcal{A}$ wins and produces a valid output.

Assume towards contradiction that $\mathcal{A}$ wins and produces a valid output but $\mathcal{B}$ loses. Then, by definition, either the outputs don't collide or the inputs are equal. First, if the outputs don't collide

$$\mathsf{F}^{2\kappa}_{K_1}(1, N_{P1}, \mathsf{ixor}(A_1, T_1)) \neq \mathsf{F}^{2\kappa}_{K_2}(1, N_{P2}, \mathsf{ixor}(A_2, T_2)))\,,$$

then the contexts produce different outer tags T^* implying that one of the decryptions fails. Thus, $\mathcal{A}$ could not have won the CMT game. Second, if the inputs are equal, then

$$(K_1, 1, N_{P1}, \mathsf{ixor}(A_1, T_1)) = (K_2, 1, N_{P2}, \mathsf{ixor}(A_2, T_2))\,,$$

which implies that both contexts have the same key $K_1 = K_2$ and public nonce $N_{P1} = N_{P2}$. This means that the only way $\mathcal{A}$ could have won the CMT game was if $A_1 \neq A_2$. But AEAD.ParDec is a deterministic function of the key, public nonce, and the ciphertext $C^* \parallel T^*$ which means that the inner tags $T_1 = T_2$. And $T_1 = T_2$ and $\mathsf{ixor}(A_1, T_1) = \mathsf{ixor}(A_2, T_2)$ implies that $A_1 = A_2$. Thus, $\mathcal{A}$ could not have won the CMT game. □

3.8.2 Secret Nonce Birthday Attack on XtH

First, we need to define some terminology. For a tag-based AEAD AEAD, let $\mathsf{TColl}_{\mathsf{AEAD}}(u, B)$ be the probability of a tag collision (for any user) when AEAD.Enc is queried for u users with B distinct encryption contexts per user. To gain some intuition for this quantity, suppose AEAD generates tags at random. That is, for each user and distinct encryption context, AEAD.Enc outputs a tag T that is uniformly random τ-bit string. Then, from Lemma 17, the probability of an inner tag collision for each user is at least

$$\frac{B(B-1)}{4 \cdot 2^\tau}.$$

So, the probability of no collision across the u users is at most

$$\left(1 - \frac{B(B-1)}{4 \cdot 2^\tau}\right)^u \leq 1 - \frac{uB(B-1)}{8 \cdot 2^\tau},$$

where the inequality uses Lemma 18. Finally, taking the event complement gives us the lower bound

$$\mathsf{TColl}_{\mathsf{AEAD}}(u, B) \geq \frac{uB(B-1)}{8 \cdot 2^\tau}.$$

Furthermore, for a given tag length τ, with a standard compression assumption, random tags have the lowest collision probability, so informally this lower bound holds for all AEADs.

```
Adversary $\mathcal{A}^{\text{NAEKG,NAEENC,NAEVER}}$
$u \leftarrow q/B$
$M \leftarrow \varepsilon$; $A \leftarrow \varepsilon$; $N_P \leftarrow 0^{n_p}$
for $i \in \{1, \dots, u\}$ :
    for $j \in \{1, \dots, B\}$ :
        $N_S \leftarrow \mathsf{encode}_{n_s}(j)$
        $C^* \parallel T \leftarrow \text{NAEENC}(i, N_P, N_S, A, M)$
        if $(T \in S_i)$ then return 1
        $S_i \leftarrow S_i \cup \{T\}$
return 0
```

```
Real NAEENC$(i, N_P, N_S, A, M)$
if $(i, N_P, N_S) \in \mathcal{Q}_N$ then return $\perp$
$L_i \leftarrow \mathsf{F}^{k}_{K_i}(1)$
$C^* \parallel T \leftarrow \mathsf{AEAD.Enc}(L_i, N_P, N_S, \varepsilon, M)$
$T^* \leftarrow \mathsf{F}^{2\kappa}_{K_i}(2, N_P, \mathsf{ixor}(A, T))$
$\mathcal{Q}_D \leftarrow \mathcal{Q}_D \cup \{(i, N_P, A, C^* \parallel T^*)\}$
$\mathcal{Q}_N \leftarrow \mathcal{Q}_N \cup \{(i, N_P, N_S)\}$
return $C^* \parallel T^*$
```

```
Random NAEENC$(i, N_P, N_S, A, M)$
if $(i, N_P, N_S) \in \mathcal{Q}_N$ then return $\perp$
$C^* \leftarrow\!\!\$\ \{0,1\}^{|M|}$
$T^* \leftarrow\!\!\$\ \{0,1\}^{2\kappa}$
$\mathcal{Q}_D \leftarrow \mathcal{Q}_D \cup \{(i, N_P, A, C^* \parallel T^*)\}$
$\mathcal{Q}_N \leftarrow \mathcal{Q}_N \cup \{(i, N_P, N_S)\}$
return $C^* \parallel T^*$
```

Figure 3.17: Definition of NAE adversary $\mathcal{A}$, used in proof of Theorem 16.

Proposition 16. *Let* AEAD *be a tag-based AEAD with key length $k \le 4\kappa$, tag length τ, public nonce length n_p, secret nonce length n_s, and associated data space $\mathcal{A} \supseteq \{\varepsilon\}$. Let* $\mathsf{F} : \{0,1\}^{2\kappa} \times \mathcal{L} \times \{0,1\}^{**} \to \{0,1\}^*$ *be a variable-output-length collision-resistant PRF with length space $\mathcal{L} = \{k, 2\kappa\}$. Then we can construct a* NAE *adversary $\mathcal{A}$ making q queries to u users and at most B queries per user, such that*

$$\mathsf{Adv}^{\mathrm{nae}}_{\mathsf{XtH[F,AEAD]}}(\mathcal{A}) \ge \mathsf{TColl}_{\mathsf{AEAD}}(q/B, B) - \frac{q(B-1)}{2 \cdot 2^{2\kappa}} ,$$

where $q < 2^{\tau/2}$, $B \le |\mathcal{N}_S|$, and $\mathsf{TColl}_{\mathsf{AEAD}}(u, B)$ *is defined above.*

Proof. The adversary $\mathcal{A}$ performs a generic multi-user birthday attack on the tag of the base AEAD scheme. This is similar to a previous such attack on block size [97, p.10]. The adversary targets $u \leftarrow q/B$ users. For each user i, it picks B distinct secret nonces N_S using a counter, and queries $\mathsf{AEAD.Enc}(i, 0^{n_p}, N_S, \varepsilon, \varepsilon)$ to get a ciphertext core C^* and an inner tag T, tracking all the inner tags T for a user i with the set S_i. If it sees the same tag again for a user, that is $T \in S_i$, it outputs 1, else it outputs 0. The pseudocode for $\mathcal{A}$ is given in Figure 3.17 and its advantage is analyzed below.

In the random world, each 2κ-bit outer tag is a uniformly random string, so from

Lemma 17, the probability of a collision for each user is at most

$$\frac{B(B-1)}{2 \cdot 2^{2\kappa}}\,.$$

Summing over u users, we get

$$\Pr[\mathrm{NAE}^{\mathrm{rand}}(\mathcal{A}) \Rightarrow 1] \leq \frac{q(B-1)}{2 \cdot 2^{2\kappa}}\,.$$

In the real world, the 2κ-bit outer tag is generated as

$$T^* \leftarrow \mathsf{F}^{2\kappa}_{K_i}(2, N_P, \mathsf{ixor}(A, T))\,, \tag{3.3}$$

where T is the τ-bit inner tag generated as

$$C^* \parallel T \leftarrow \mathsf{AEAD.Enc}(L_i, N_P, N_S, \varepsilon, M)\,,$$

where L_i is a subkey generated from the user's key K_i. Bad be the event that $T \in S_i$, that is there's an inner tag collision. Since the adversary $\mathcal{A}$ keeps the public nonce N_P, associated data A, and message M, constant across all queries, from Equation (3.3) Bad also implies that the corresponding 2κ-bit outer tags also collide. This implies that

$$\Pr[\mathrm{NAE}^{\mathrm{real}}(\mathcal{A}) \Rightarrow 1] \geq \Pr[\mathrm{NAE}^{\mathrm{real}}(\mathcal{A}) \text{ causes } \mathsf{Bad}] \geq \mathsf{TColl}_{\mathsf{AEAD}}(q/B, B)\,,$$

where $\mathsf{TColl}_{\mathsf{AEAD}}(u, B)$ is a lower bound on the probability of AEAD outputting the same inner tag when queried for u users with B distinct encryption contexts per user. □

Lemma 17 (Proposition A.1 in [84]). *Let $C_n(q)$ be the probability of a collision when we pick q n-bit strings uniformly at random. Then, it holds that*

$$C_n(q) \leq \frac{q(q-1)}{2 \cdot 2^n}\,,$$

and if $q \leq 2^{n/2}$, it also holds that

$$C_n(q) \geq \frac{q(q-1)}{4 \cdot 2^n} \, .$$

Lemma 18 (Lemma 3.3 from [97])**.** *Let $p \geq 1$ be an integer and $a \geq 0$ a real number. Then, if $ap \leq 1$, it holds that $(1-a)^p \leq 1 - \frac{ap}{2}$.*

3.8.3 NAX does not imply NAE

Consider the CTR-then-AXU scheme with no nonce hiding, where the ciphertext C is a CTR mode encryption of the plaintext, and the tag is an AXU hash of the ciphertext $T \leftarrow \mathsf{G}_H(C)$, where the AXU hash key H is derived from the main key.

This scheme is not NAE secure, and we can construct an efficient adversary $\mathcal{A}$ as follows. Adversary $\mathcal{A}$ encrypts the empty message ε twice, under different public nonces, and outputs 1 if the tags collide and 0 otherwise. In the real game, since the tag is only a function of the ciphertext (which is always empty in this case) and the key, it produces the same tag twice. But, in the random game, since the tag is randomly generated, it produces different tags with high probability. Therefore, the adversary makes two NAEEnc queries and distinguishes with high probability.

However, this scheme is NAX secure since the producing a tag collision for the same public nonce (as required by NAX), requires either breaking CTR mode to produce identical ciphertexts or breaking the AXU hash to produce a collision on distinct messages.

```
Adversary $\mathcal{B}^{\mathrm{NAEKG},\ldots}$

bad ← false
$b \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{pNaxKg},\ldots}$
if bad = true then
    return true
return $b$

Procedure pNaxKg()

NAEKg()
```

```
Procedure pNaxEnc($i, N_P, N_S, A, M, X$)

if $(i, N_P, N_S) \in Q_N$ then return $\perp$
$C^* \,\|\, T \leftarrow$ NAEEnc($i, N_P, N_S, A, M$)
$Q_D \leftarrow Q_D \cup \{(i, N_P, A, C^*, X)\}$
$Q_N \leftarrow Q_N \cup \{(i, N_P, N_S)\}$
$Q'_D \leftarrow Q'_D \cup \{(i, N_P, A, C^* \,\|\, T)\}$
win ← $(T \oplus X \in S_{i,N_P})$
if win = true then
    bad ← true
$S_{i,N_P}[N_S] \leftarrow T \oplus X$
return $C^*$, win
```

```
Procedure pNaxVer($i, N_P, {N_S}^*, A, C^*, X$)

if $(i, N_P, A, C^*, X) \in Q_D$ then
    return $\perp$
if $(i, N_P, {N_S}^*) \notin Q_N$ then
    return false
$T \leftarrow X \oplus S_{i,N_P}[{N_S}^*]$
if $(i, N_P, A, C^* \,\|\, T) \in Q'_D$ then
    bad ← true
    return false
return NAEVer($i, N_P, A, C^* \,\|\, T$)
```

Figure 3.18: Definition of NAE adversary $\mathcal{B}$ used in proof of Proposition 10. $\mathcal{B}$ is given access to NAE oracles NAEKg, NAEEnc, NAEVer (defined in Figure 3.4) corresponding to a tag-based AEAD scheme AEAD. It constructs NAX oracles pNaxKg, pNaxEnc, pNaxVer (defined in Figure 3.7) for the same AEAD scheme AEAD, and provides them to the NAX adversary $\mathcal{A}$.

3.8.4 Proof of Proposition 10

Adversary $\mathcal{B}$ maintains a flag bad that is initialized to **false** and simulates the game $\mathrm{NAX}^{\mathrm{real}}(\mathcal{A})$ with the following changes. For each NaxEnc query, instead of calling AEAD.Enc, it queries its own NAEEnc. If this results in win = **true**, then it sets bad ← **true**. For each NaxVer query $(i, N_P, {N_S}^*, A, C^*, X)$, if there is no previous NaxEnc query to the same user i and nonce (N_P, N_S), then it simply returns **false**. Suppose there was a previous NaxEnc query $(i, N_P, {N_S}^*, A^*, M^*, X^*)$ that led to NAEEnc query $(i, N_P, {N_S}^*, A^*, M^*)$ whose tag is T^*. Then it sets $T \leftarrow T^* \oplus X^* \oplus X$ and calls NAEVer$(i, N_P, A, C^* \,\|\, T)$. If this query does not violate the definitional restriction, then it returns the answer, otherwise it sets bad ← **true** and returns **false**. Adversary $\mathcal{B}$ is defined formally in Figure 3.18 and analyzed below.

In the real game $\mathrm{NAE}^{\mathrm{real}}(\mathcal{B})$, if bad stays **false** then it perfectly simulates the real game $\mathrm{NAX}^{\mathrm{real}}(\mathcal{A})$, and if bad is set to **true**, then it always returns **true**, so we get

$$\Pr[\mathrm{NAE}^{\mathrm{real}}(\mathcal{B}) \Rightarrow \textbf{true}] \geq \Pr[\mathrm{NAX}^{\mathrm{real}}(\mathcal{A}) \Rightarrow \textbf{true}]\,.$$

Similarly, in the random game $\mathrm{NAE}^{\mathrm{rand}}(\mathcal{B})$, if bad stays **true**, then it perfectly simulates the random game $\mathrm{NAX}^{\mathrm{rand}}(\mathcal{A})$, thus

$$\Pr[\mathrm{NAE}^{\mathrm{rand}}(\mathcal{B}) \Rightarrow \textbf{true}] \leq \Pr[\mathrm{NAX}^{\mathrm{rand}}(\mathcal{A}) \Rightarrow \textbf{true}] + \Pr[\mathrm{NAE}^{\mathrm{rand}}(\mathcal{B}) \text{ sets } \mathsf{bad}]\,.$$

We now bound the probability that $\mathrm{NAE}^{\mathrm{rand}}(\mathcal{B})$ sets bad to **true**.

First, consider a $\textsc{NaxEnc}$ query (i, N_P, N_S, A, M, X). For each choice of user i and public nonce N_P, it can make at most $\min(B, 2^{n_s})$ queries, and each query produces a uniformly random τ-bit tag. Thus, the probability that this query sets $\mathsf{bad} \leftarrow \textbf{true}$ is

$$\Pr[T \oplus X \in S_{i,N_P}] \leq \frac{\min(B, 2^{n_s})}{2^{\tau}}\,.$$

Next, consider a $\textsc{NaxVer}$ query $(i, N_P, N_S{}^*, A, C^*, X)$ with the corresponding previous $\textsc{NaxEnc}$ query $(i, N_P, N_S{}^*, A^*, M^*, X^*)$. Let the $\mathrm{NAE}\textsc{Enc}(i, N_P, N_S{}^*, A^*, M^*)$ be the internal query whose tag is T^*. Let $T \leftarrow T^* \oplus X^* \oplus X$. For this query to set $\mathsf{bad} \leftarrow \textbf{true}$, $\mathcal{B}$ must have previously queried $\mathrm{NAE}\textsc{Enc}(i, N_P, N_S, A, M) \rightarrow C^* \parallel T$. We proceed by case analysis.

Case 1: $N_S = N_S{}^*$. Since we already know both queries used the same public nonce N_P, nonce uniqueness implies that $A = A^*$ and $T = T^*$. This further implies that $X = X^*$. Thus, $\mathcal{A}$ violates its definitional restriction by querying $\textsc{NaxEnc}(i, N_P, N_S, A, M^*, X)$ to get C^*, and then querying $\textsc{NaxVer}(i, N_P, N_S, A, C^*, X)$. So, $\mathsf{bad} \leftarrow \textbf{true}$ could not have been set this way.

Case 2: $N_S \neq N_S{}^*$. For any query $\mathrm{NAE}\textsc{Enc}(i, N_P, N_S, A, M) \rightarrow C^* \parallel T$ with $N_S \neq N_S{}^*$, the produced tag T is a uniformly random τ-bit string, independent of T^*. And it could have made at most $\min(B, 2^{n_s})$ such queries. Thus, the probability that this query sets $\mathsf{bad} \leftarrow \textbf{true}$ is

$$\Pr[T \oplus X \in S_{i,N_P}] \leq \frac{\min(B, 2^{n_s})}{2^\tau}\,.$$

Finally, summing over q queries in total, the probability that $\mathsf{bad} \leftarrow \textbf{true}$ is set in $\mathrm{NAE}^{\mathrm{rand}}(\mathcal{B})$ is at most

$$\Pr[\mathrm{NAE}^{\mathrm{rand}}(\mathcal{B}) \text{ sets } \mathsf{bad}] \leq \frac{q \cdot \min(B, 2^{n_s})}{2^\tau}\,,$$

and we get the claimed bound by subtracting output probabilities. □

3.8.5 Variants of CTY and XtH

In Figure 3.19, we define CTZ and YtH. CTZ is a variant of CTY that generates a subkey for the base AEAD and therefore can be proved secure in the standard model. YtH is a variant of XtH that hashes the secret nonce, this trades off XtH's requirement that the base AEAD have a tag of length at least $\min(\kappa + n_s, 2\kappa)$ for hashing the secret nonce.

3.8.6 NAE security of XtH

We first prove the security of XtH assuming an idealized CR-PRF in Lemma 19, then use that to prove Theorem 11.

Let κ be the security parameter (we will later set it to 128 bits.) For a finite key space $\mathcal{K}$ and a finite set X, we use $\mathrm{keyPerms}(\mathcal{K}, X)$ to denote the set of all families of permutations $\Pi_K : X \to X$ indexed by a key $K \in \mathcal{K}$. For a finite key space $\mathcal{K}$ and finite

```
$\mathsf{YtH[F, AEAD].Enc}(K, N_P, N_S, A, M)$
    $L \leftarrow \mathsf{F}^k_K(1)$
    $C^* \parallel T \leftarrow \mathsf{AEAD.Enc}(L, N_P, N_S, \varepsilon, M)$
    $T^* \leftarrow \mathsf{F}^{2\kappa}_K(2, N_P, N_S, \mathsf{ixor}(A, T))$
    return $C^* \parallel T^*$

$\mathsf{YtH[F, AEAD].Dec}(K, N_P, A, C^* \parallel T^*)$
    $L \leftarrow \mathsf{F}^k_K(1)$
    $(M, N_S, T) \leftarrow \mathsf{AEAD.ParDec}(L, N_P, \varepsilon, C^*)$
    $T' \leftarrow \mathsf{F}^{2\kappa}_K(2, N_P, N_S, \mathsf{ixor}(A, T))$
    if $T' = T^*$ then return $(M, N_S)$
    else return $\perp$

$\mathsf{ixor}(A, T)$
    $\ell \leftarrow \max(|A| + 1, |T|)$
    $X \leftarrow (A \parallel 10^{\ell-|A|-1})$
        $\oplus (T \parallel 0^{\ell-|T|})$
    return $X$

$\mathsf{CTZ[F, AEAD].Enc}(K, N_P, N_S, A, M)$
    $L \leftarrow \mathsf{F}^k_K(1)$
    $C^* \parallel T \leftarrow \mathsf{AEAD.Enc}(L, N_P, N_S, \varepsilon, M)$
    $T^* \leftarrow \mathsf{F}^{2\kappa}_K(1, N_P, N_S, A, T)$
    return $C^* \parallel T^*$

$\mathsf{CTZ[F, AEAD].Dec}(K, N_P, A, C^* \parallel T^*)$
    $L \leftarrow \mathsf{F}^k_K(1)$
    $(M, N_S, T) \leftarrow \mathsf{AEAD.ParDec}(L, N_P, \varepsilon, C^*)$
    $T' \leftarrow \mathsf{F}^{2\kappa}_K(1, N_P, N_S, A, T)$
    if $T' = T^*$ then return $(M, N_S)$
    else return $\perp$
```

Figure 3.19: **Definitions of** $\mathsf{YtH[F, AEAD]}$ **and** $\mathsf{CTZ[F, AEAD]}$**.** Here, $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{ParDec})$ is a tag-based AEAD with key length k, and $\mathsf{F} : \{0,1\}^{2\kappa} \times \mathcal{L} \times \{0,1\}^{**} \to \{0,1\}^*$ is a variable-output-length collision-resistant PRF with length space $\mathcal{L} = \{k, 2\kappa\}$.

sets X and Y, we use keyFuncs$(\mathcal{K}, X, Y)$ to denote the set of all families of functions $f_K : X \to Y$ indexed by a key $K \in \mathcal{K}$.

Lemma 19. *Let $\mathcal{K}_{\mathsf{XtH}} := \{0,1\}^{2\kappa}$ and $\mathcal{A}_{\mathsf{XtH}}$ be the key space and associated data space for* XtH*, respectively. Let* AEAD *be a tag-based AEAD with key length k, tag length τ, public nonce length n_p, and secret nonce length n_s. Define the* ixor *output space $\mathcal{Y}_{\mathsf{ixor}} := \{\mathsf{ixor}(A, T) : A \in \mathcal{A}_{\mathsf{XtH}}, T \in \{0,1\}^\tau\}$. Let $f_i \leftarrow_\$ \text{keyFuncs}(\{0,1\}^{2\kappa}, \{0,1\}^{n_p} \times \mathcal{Y}_{\mathsf{ixor}}, \{0,1\}^{2\kappa})$ and $g_i \leftarrow_\$ \text{keyFuncs}(\{0,1\}^{2\kappa}, \{\varepsilon\}, \{0,1\}^k)$ be a uniformly random function. Let $\mathcal{A}$ be an orderly NAE adversary making q queries, with q_v verification queries. Assume $\mathcal{A}$ queries u users, making at most B queries (encryption and verification) per user. Then, we can construct an orderly NAX adversary C such that*

$$\mathsf{Adv}^{\mathrm{nae}}_{\mathsf{XtH}[(f_i, g_i), \mathsf{AEAD}]}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{nax}}_{\mathsf{AEAD}}(C) + \frac{2qB - q}{2^{2\kappa}}.$$

Adversary C makes at most q NAX queries, with at most q_v verification queries, to u users, making at most B queries (encryption and verification) per user, and has running time similar to $\mathcal{A}$.

```
Games $G_1(\mathcal{A})$, $\boxed{G_2(\mathcal{A})}$

Procedure $\text{Kg}()$
$u \leftarrow u+1;\ L_u \leftarrow\$ \{0,1\}^k$

Procedure $\text{NAEEnc}(i, N_P, N_S, A, M)$
if $(i, N_P, N_S) \in Q_N$ then return $\perp$
$C^* \parallel T \leftarrow \mathsf{AEAD.Enc}(L_i, N_P, N_S, \varepsilon, M)$
$T^* \leftarrow \text{EncTag}(i, N_P, \mathsf{ixor}(A, T))$
$Q_D \leftarrow Q_D \cup \{(i, N_P, A, C)\}$
$Q_N \leftarrow Q_N \cup \{(i, N_P, N_S)\}$
return $C^* \parallel T^*$

Procedure $\text{NAEVer}(i, N_P, A, C^* \parallel T^*)$
if $(i, N_P, A, C) \in Q_D$ then return $\perp$
$(M, N_S, T) \leftarrow \mathsf{AEAD.ParDec}(L_i, N_P, \varepsilon, C^*)$
$T' \leftarrow \text{VerTag}(i, N_P, \mathsf{ixor}(A, T))$
if $T' \neq T^*$ then return false
return true

Procedure $\text{EncTag}(i, X)$
if $\mathsf{Tbl}[i, X] \neq \perp$ then return $\mathsf{Tbl}[i, X]$
$Y \leftarrow\$ \{0,1\}^{2\kappa}$
if $Y \in S_i$ then
    $\mathsf{bad}_0 \leftarrow$ true; $\boxed{Y \leftarrow\$ \{0,1\}^{2\kappa} \setminus S_i}$
$S_i \leftarrow S_i \cup \{Y\}$
$\mathsf{Tbl}[i, X] \leftarrow Y$
return $\mathsf{Tbl}[i, X]$

Procedure $\text{VerTag}(i, X)$
if $\mathsf{Tbl}[i, X] = \perp$
    $\mathsf{Tbl}[i, X] \leftarrow\$ \{0,1\}^{2\kappa}$
return $\mathsf{Tbl}[i, X]$
```

```
Games $\boxed{G_3(\mathcal{A})}$, $G_4(\mathcal{A})$

Procedure $\text{Kg}()$
$u \leftarrow u+1$

Procedure $\text{NAEEnc}(i, N_P, N_S, A, M)$
if $(i, N_P, N_S) \in Q_N$ then return $\perp$
$C^* \leftarrow\$ \{0,1\}^{|M|}$
$T^* \leftarrow \text{EncTag}(i)$
$Q_D \leftarrow Q_D \cup \{(i, N_P, A, C)\}$
$Q_N \leftarrow Q_N \cup \{(i, N_P, N_S)\}$
return $C^* \parallel T^*$

Procedure $\text{NAEVer}(i, N_P, A, C^* \parallel T^*)$
if $(i, N_P, A, C^* \parallel T^*) \in Q_D$ then
    return $\perp$
return false

Procedure $\text{EncTag}(i)$
$Y \leftarrow\$ \{0,1\}^{2\kappa}$
if $Y \in S_i$ then
    $\mathsf{bad}_0 \leftarrow$ true; $\boxed{Y \leftarrow\$ \{0,1\}^{2\kappa} \setminus S_i}$
$S_i \leftarrow S_i \cup \{Y\}$
return $Y$
```

Figure 3.20: Games G_1-G_4 used in proof of Theorem 11. Games G_2 and G_3 include boxed statements.

Proof. We are going to use a sequence of games to transition from NAE^{real} to NAE^{rand}. Game G_0 is the real game NAE^{real}, and games G_1 to G_4 are given in Figure 3.20.

Game G_1 is identical to game G_0 except it emulates the random function g_i by sampling subkeys at random and emulates f_i with new procedures EncTag and VerTag. Since each random function $g_i : \{\varepsilon\} \to \{0,1\}^k$ is only called once with fixed ε input, it is equivalent to randomly sampling an output. The random functions f_i are invoked in

```
Adversary $C^{\textsc{NaxKg},\ldots}$

$b \leftarrow\$ \mathcal{A}^{\textsc{pNAEKg},\textsc{pNAEEnc},\textsc{pNAEVer}}$
return $b$

Procedure pNAEKg()

NaxKg()

Function prefix($A$)

if $|A| \geq \tau$ then
    return $A[1 : \tau]$
return $A \parallel 10^{\tau-|A|-1}$
```

```
Procedure pNAEEnc($i, N_P, N_S, A, M$)

if $(i, N_P, N_S) \in \mathcal{Q}_N$ then return $\perp$
$X \leftarrow \mathsf{prefix}(A)$
$(C^*, \mathsf{win}) \leftarrow \textsc{NaxEnc}(i, N_P, N_S, \varepsilon, M, X)$
if $\mathsf{win} = \mathbf{true}$ then
    terminate and output 1
$T^* \leftarrow\$ \{0,1\}^{2\kappa} \setminus S_i$; $S_i \leftarrow S_i \cup \{T^*\}$
$W_i[T^*] \leftarrow N_S$
$Z_i[T^*] \leftarrow (C^*, X)$
$\mathcal{Q}_D \leftarrow \mathcal{Q}_D \cup \{(i, N_P, A, C^* \parallel T^*)\}$
$\mathcal{Q}_N \leftarrow \mathcal{Q}_N \cup \{(i, N_P, N_S)\}$
return $C^* \parallel T^*$
```

```
Procedure pNAEVer($i, N_P, A, C^* \parallel T^*$)

if $(i, N_P, A, C^* \parallel T^*) \in \mathcal{Q}_D$ then
    return $\perp$
if $T^* \notin S_i$ then
    return false
$X' \leftarrow \mathsf{prefix}(A)$
$N_S \leftarrow W_i[T^*]$
if $(C^*, X') \neq Z_i[T^*]$ then
    $\mathsf{ret} \leftarrow \textsc{NaxVer}(i, N_P, N_S, \varepsilon, C^*, X')$
    if $\mathsf{ret} = \mathbf{true}$ then
        terminate and output 1
return false
```

Figure 3.21: Definition of NAX adversary $\mathcal{C}$ used in proof of Theorem 11. $\mathcal{C}$ is given access to NAX oracles NaxKg, NaxEnc, NaxVer corresponding to a tag-based AEAD scheme AEAD. It constructs NAE oracles pNAEKg, pNAEEnc, pNAEVer for the same AEAD scheme AEAD, and provides them to the orderly NAE adversary $\mathcal{A}$.

NAEEnc and NAEVer to compute tags, it can be emulated using a table Tbl. Thus,

$$\Pr[G_0(\mathcal{A})] - \Pr[G_1(\mathcal{A})] = 0\,.$$

Game G_2 is identical to game G_1, except the tags T^* of encryption queries for the same user are distinct. It does this by tracking, for each user i, the values generated inside EncTag$(i, \cdot)$ using a set S_i. And if value reappears, it sets bad_0 to **true** and re-samples a fresh output from $\{0,1\}^{2\kappa} \setminus S_i$. Since $\mathcal{A}$ makes at most B queries per user the probability that a given query leads to a collision is at most $(B-1)/2^{2\kappa}$, and applying the union bound over at most q EncTag queries and using the fact that G_1 and G_2 only differ when $\mathsf{bad}_0 \leftarrow \mathbf{true}$, we get

$$\Pr[G_1(\mathcal{A})] - \Pr[G_2(\mathcal{A})] \leq \Pr[G_2(\mathcal{A}) \text{ sets } \mathsf{bad}_0] \leq \frac{q(B-1)}{2^{2\kappa}}\,.$$

Game G_3 is the random game $\mathrm{NAE}^{\mathrm{rand}}$ except tags T^* for encryption queries of the same user are distinct like G_2. Recall that if $\mathsf{ixor}(A, T) = \mathsf{ixor}(A', T')$, then $\mathsf{prefix}(A) \oplus$

$T = \mathsf{prefix}(A') \oplus T'$. Now, we are going to bound the gap between G_2 and G_3 by constructing an NAX adversary $\mathcal{C}$ for AEAD which emulates $\mathcal{A}'$. For each encryption query (i, N_P, N_S, A, M) of $\mathcal{A}'$, adversary $\mathcal{C}$ queries $(C^*, \mathsf{win}) \leftarrow \textsc{NaxEnc}(i, N_P, N_S, \varepsilon, M, X)$, where $X \leftarrow \mathsf{prefix}(A)$. If $\mathsf{win} = \mathbf{true}$, then it terminates and outputs 1. Otherwise, it computes $T^* \leftarrow\$ \{0,1\}^{2\kappa} \setminus S_i$, adds T^* to S_i, and returns $C^* \parallel T^*$ to $\mathcal{A}$. For each verification query $(i, N_P, A, C^* \parallel T^*)$ of $\mathcal{A}$, adversary $\mathcal{C}$ always returns $\mathbf{false}$. Still, if (i) there's a prior query $(C_e^*, \mathsf{win}) \leftarrow \textsc{NaxEnc}(i, N_P, N_S, \varepsilon, M, X_e)$ of $\mathcal{C}$ that led to same tag T^* (since tags are unique, there's at most one such query) and (ii) $(C_e^*, X_e) \neq (C^*, X')$ where $X' \leftarrow \mathsf{prefix}(A)$, then $\mathcal{C}$ queries $\textsc{NaxVer}(i, N_P, N_S, A, C^*, X')$; if this query returns $\mathbf{true}$, then it terminates and outputs 1. By construction, adversary $\mathcal{C}$ does not violate definitional restrictions, and if it doesn't terminate prematurely, then it outputs the same answer as $\mathcal{A}$. Adversary $\mathcal{C}$ is defined with pseudocode in Figure 3.21 and analyzed below.

To analyze the advantage of $\mathcal{C}$, consider the real game $\mathrm{NAX}^{\mathrm{real}}(\mathcal{C})$. Let bad_1 be the event that (i) $\mathcal{C}$ does not terminate prematurely and (ii) there is a verification query of $\mathcal{A}$ that is supposed to return $\mathbf{true}$, but $\mathcal{C}$ returns $\mathbf{false}$. If bad_1 doesn't happen, then $\mathcal{C}$ either correctly simulates $\mathrm{G}_2(\mathcal{A})$ or prematurely terminates and outputs 1. Thus,

$$\Pr[\mathrm{NAX}^{\mathrm{real}}(\mathcal{C}) \Rightarrow 1] \geq \Pr[\mathrm{G}_2(\mathcal{A}') \Rightarrow 1] - \Pr[\mathsf{bad}_1]\,.$$

To bound the probability of bad_1, consider a NAEVer query $(i, N_P, A, C^* \parallel T^*)$ of $\mathcal{A}$, and let $X' \leftarrow \mathsf{prefix}(A)$. Let T be the tentative internal tag that $\mathsf{AEAD.ParDec}$ would return on processing ciphertext core C^* and public nonce N_P, with empty associated data, for user i. Then this NAEVer query returns $\mathbf{true}$ only if

$$T^* = \textsc{VerTag}(i, N_P, \mathsf{ixor}(A, T))\,.$$

If $(i, N_P, \mathsf{ixor}(A, T))$ had not been queried to $\textsc{EncTag}$ or $\textsc{VerTag}$ before, then $\textsc{VerTag}$

returns a uniformly random string, independent of what $\mathcal{A}$ sees, so the chance this causes bad_1 to happen is at most $2^{-2\kappa}$. Suppose $(i, N_P, \mathsf{ixor}(A, T))$ was queried to EncTag or VerTag before. Since $\mathcal{A}$ is orderly and makes all its verification queries at once, it must have been queried inside an encryption query. Furthermore, by construction NAEEnc must have been queried for the same user i, with the same public nonce N_P, with some secret nonce N_S, but potentially with a different associated data A', and different message M'. Let this query $\mathcal{A}$ makes be $\text{NAEEnc}(i, N_P, N_S, A', M') \to C^*_e \parallel T^*_e$ with inner tag T_e. We proceed by a case-analysis.

Case 1: the NAEEnc query produces a different outer tag $T^*_e \neq T^*$, then $T^* \neq \text{VerTag}(i, N_P, \mathsf{ixor}(A, T))$ and this verification query does not cause bad_1.

Case 2: the NAEEnc query produces the same outer tag $T^*_e = T^*$ but either was queried with a different associated data prefix or it returns a different ciphertext core, i.e., $(C^*_e, \mathsf{prefix}(A')) \neq (C^*, \mathsf{prefix}(A))$. Then, in its verification query, $\mathcal{C}$ queries NaxVer with $(i, N_P, N_S, \varepsilon, C^*, \mathsf{prefix}(A))$. The winning condition $\mathsf{ixor}(A, T) = \mathsf{ixor}(A', T_e)$ implies the NaxVer winning condition $\mathsf{prefix}(A) \oplus T = \mathsf{prefix}(A') \oplus T_e$, and this is a valid NaxVer query since it uses a different ciphertext core or associated data. So, NaxVer returns **true** and bad_1 does not happen.

Case 3: the NAEEnc query produces the same outer tag $T^*_e = T^*$ and was queried with the same associated data prefix and it returns the same ciphertext core, i.e., $(C^*_e, \mathsf{prefix}(A')) = (C^*, \mathsf{prefix}(A))$. The winning condition $\mathsf{ixor}(A, T) = \mathsf{ixor}(A', T_e)$ implies that $\mathsf{prefix}(A) \oplus T = \mathsf{prefix}(A') \oplus T_e$. Combined with $\mathsf{prefix}(A) = \mathsf{prefix}(A')$, this gives us that the internal tags are equal $T = T_e$. But, if $T = T_e$ then by definition $\mathsf{ixor}(A, T) = \mathsf{ixor}(A', T_e)$ implies that the associated data are equal $A = A'$. This means that $\mathcal{A}$'s NAEVer query is invalid, since it is querying a previously returned $C^* \parallel T^*$ with the same public nonce N_P and the same associated data A. So, bad_1 does not happen.

Hence, each verification query can cause bad_1 with probability at most $2^{-2\kappa}$. Summing over q queries $\Pr[\mathsf{bad}_1] \le q \cdot 2^{-2\kappa}$, and thus

$$\Pr[\mathrm{NAX}^{\mathrm{real}}(\mathcal{C}) \Rightarrow 1] \ge \Pr[\mathrm{G}_2(\mathcal{A}) \Rightarrow 1] - \frac{q}{2^{2\kappa}} .$$

Next, in the random world, $\mathcal{C}$ perfectly simulates game G_3, and thus

$$\Pr[\mathrm{NAX}^{\mathrm{rand}}(\mathcal{C}) \Rightarrow 1] = \Pr[\mathrm{G}_3(\mathcal{A}) \Rightarrow 1] .$$

Hence,

$$\Pr[\mathrm{G}_2(\mathcal{A})] - \Pr[\mathrm{G}_3(\mathcal{A})] \le \mathsf{Adv}^{\mathrm{nax}}_{\mathsf{AEAD}}(\mathcal{C}) + \frac{q}{2^{2\kappa}} .$$

Game G_4 is the random game $\mathrm{NAE}^{\mathrm{rand}}$, and games G_3 and G_4 only differ on whether tags T^* for encryption queries of the same user are distinct, and thus

$$\Pr[\mathrm{G}_3(\mathcal{A})] - \Pr[\mathrm{G}_4(\mathcal{A})] \le \Pr[\mathrm{G}_3(\mathcal{A}) \text{ sets } \mathsf{bad}_0] \le \frac{q(B-1)}{2^{2\kappa}} .$$

Summing up,

$$\mathsf{Adv}^{\mathrm{nae}}_{\mathsf{XtH[F,AEAD]}}(\mathcal{A}) \le \Pr[\mathrm{G}_0(\mathcal{A})] - \Pr[\mathrm{G}_4(\mathcal{A})] \le \mathsf{Adv}^{\mathrm{nax}}_{\mathsf{AEAD}}(\mathcal{C}) + \frac{2qB - q}{2^{2\kappa}} .$$

This completes the proof. □

Proof of Theorem 11. First, let's use Theorem 8 on $\mathcal{A}$ to construct an orderly NAE adversary $\mathcal{A}'$ such that

$$\mathsf{Adv}^{\mathrm{nae}}_{\mathsf{XtH[F,AEAD]}}(\mathcal{A}) \le \mathsf{Adv}^{\mathrm{nae}}_{\mathsf{XtH[F,AEAD]}}(\mathcal{A}') + \frac{q}{2^{2\kappa}} ,$$

where 2κ is the ciphertext overhead of XtH and adversary $\mathcal{A}'$ makes at most q queries, with at most q_v verification queries, to u users and at most B queries per user, and has running time similar to $\mathcal{A}$. Now, consider the following sequence of games.

Game G_0 is the real game $\mathrm{NAX}^{\mathrm{real}}$, game G_1 is identical except it replaces calls to the real PRF F with calls to two uniformly random functions $f_i \leftarrow\$ \mathrm{keyFuncs}(\{0,1\}^{2\kappa}, \{0,1\}^{n_p} \times \mathcal{Y}_{\mathsf{ixor}}, \{0,1\}^{2\kappa})$ and $g_i \leftarrow\$ \mathrm{keyFuncs}(\{0,1\}^{2\kappa}, \{\varepsilon\}, \{0,1\}^{k})$, where $\mathcal{Y}_{\mathsf{ixor}} := \{\mathsf{ixor}(A, T) : A \in \mathcal{A}_{\mathsf{XtH}},\ T \in \{0,1\}^{\tau}\}$ is the ixor output space. We bound the gap between these games by constructing a PRF adversary $\mathcal{B}$ for F that emulates $\mathrm{G}_0(\mathcal{A})$ but runs PRF game's Kg for each user and replaces calls to the F to the PRF game's Func. Then

$$\Pr[\mathrm{G}_0(\mathcal{A}')] - \Pr[\mathrm{G}_1(\mathcal{A}')] \leq \mathsf{Adv}^{\mathrm{tprp}}_{\tilde{E}}(\mathcal{B}),$$

and $\mathcal{B}$ makes at most q PRF queries to u users with at most B queries per user, and has running time similar to $\mathcal{A}$.

Game G_2 is the random game $\mathrm{NAE}^{\mathrm{rand}}$. Since the $\mathrm{NAE}^{\mathrm{rand}}$ game does not depend on the choice of PRF; that is, it produces the same output for a real PRF and an ideal PRF, we can apply Lemma 19 to get

$$\Pr[\mathrm{G}_2(\mathcal{A}')] - \Pr[\mathrm{G}_1(\mathcal{A}')] = \mathsf{Adv}^{\mathrm{nae}}_{\mathsf{XtH}[(f_i,g_i),\mathsf{AEAD}]}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{nax}}_{\mathsf{AEAD}}(\mathcal{C}) + \frac{2qB - q}{2^{2\kappa}}.$$

Subtracting differences finishes the proof. □

3.9 NAE security of ΘCH

We prove NAE security of $\mathsf{\Theta CH}$ in two stages. We start by proving the NAX security of $\mathsf{\Theta CX}$ Section 3.9.1. Then building off that result and Lemma 19 proving the NAE security of XtH, in Section 3.9.2, we prove Theorem 13.

Let κ be the security parameter (we will later set it to 128 bits.) Let $w \geq 2\kappa$ be the blocksize of the underlying TBC. Define the tweakspace for $\Theta\mathsf{CH}$ to be

$$\mathcal{T}_{\Theta\mathsf{CH}} := \mathcal{N} \times \{0, 1, \ldots, \mathsf{maxBlocks}\} \times \{\varepsilon, \star, \$, \star\$\} \tag{3.4}$$

where we can equivalently view $\{\varepsilon, \star, \$, \star\$\}$ as $\{0, 1, 2, 3\}$.

3.9.1 NAX Security of $\Theta\mathsf{CX}$

We show that $\Theta\mathsf{CX}[\tilde{E}, \mathsf{G}]$ is NAX secure, first assuming an ideal TBC in Lemma 20, then extending it to any TBC in Theorem 21.

Lemma 20. *Let $\tilde{\pi} \leftarrow_\$ \widetilde{\mathrm{Perms}}(w, \mathcal{T}_{\Theta\mathsf{CH}})$ be an ideal w-bit TBC with the $\Theta\mathsf{CH}$ tweakspace $\mathcal{T}_{\Theta\mathsf{CH}}$. Let $\mathsf{G} : \{0,1\}^h \times \{0,1\}^* \to \{0,1\}^\tau$ be an ϵ-AXU hash function. Let $\mathcal{A}$ be an NAX adversary making at most $q < 2^{2\kappa-1}$ queries, with at most q_v verification queries. Assume $\mathcal{A}$ queries u users, making at most B queries (encryption and verification) per user. Then, we have*

$$\mathsf{Adv}^{\mathrm{nax}}_{\Theta\mathsf{CX}[\tilde{\pi},\mathsf{G}]}(\mathcal{A}) \leq \frac{q \cdot \min(B, 2^{n_s})}{2^w} + \frac{2q_v}{2^\kappa} + q_v \cdot \epsilon(\ell) + q \cdot \min(B, 2^{n_s}) \cdot \epsilon(\ell)\,,$$

where $\ell := 8 + n_s + w + r$ is the maximum length that $\Theta\mathsf{CX}$ calls the AXU hash G on.

Proof. Consider the following sequence of games. Game G_0 is the real game $\mathrm{NAX}^{\mathrm{real}}$. Game G_1 is identical to game G_0 with the following changes. It maintains a flag bad that is initialized to **false**. For each encryption query, if win = **true**, then it sets $\mathsf{bad} \leftarrow$ **true** and returns **false** for the win boolean. Likewise, for each verification query, if its **true**, then it sets $\mathsf{bad} \leftarrow$ **true** and returns **false**. By construction, G_0 and G_1 are identical-until-bad so

$$\Pr[\mathrm{G}_0(\mathcal{A}) \Rightarrow 1] - \Pr[\mathrm{G}_1(\mathcal{A}) \Rightarrow 1] \leq \Pr[\mathrm{G}_1(\mathcal{A}) \text{ sets } \mathsf{bad}]\,.$$

We now bound the chance that G_1 sets $\mathsf{bad} \leftarrow \mathbf{true}$. Without loss of generality, let's assume that we terminate the game immediately when $\mathsf{bad} \leftarrow \mathbf{true}$ is set and return 1. This ensures that the AXU hash keys are independent of what $\mathcal{A}$ receives, and are independent of $\mathcal{A}$'s queries.

First, consider an encryption query (i, N_P, N_S, M, X) of $\mathcal{A}$, and let C be the corresponding ciphertext core, and let $T \leftarrow \mathsf{G}_H(N_S, \mathrm{Chk}, \mathsf{mlen})$ be the secret tag; for readability, we omitted the one-byte domain separation prefix 6. For this encryption query to set $\mathsf{bad} \leftarrow \mathbf{true}$, there must be a prior encryption query (i, N_P, N_S', M', X') with corresponding ciphertext core C' and secret tag $T' \leftarrow \mathsf{G}_H(N_S', \mathrm{Chk}', \mathsf{mlen}')$ such that $T \oplus X = T' \oplus X'$. Since G is ϵ-AXU and nonce uniqueness implies $N_S \neq N_S'$, we get that this sets $\mathsf{bad} \leftarrow \mathbf{true}$ with probability at most $\epsilon(\ell)$ where $\ell := 8 + n_s + w + r + b$ is the maximum length the AXU hash G is queried on (8 is from the one-byte domain separation prefix 6). Summing this over the maximum $\min(B, 2^{n_s})$ encryption queries to the same user i and public nonce N_P, we get

$$\Pr[\text{encryption query sets } \mathsf{bad}] \leq \min(B, 2^{n_s}) \cdot \epsilon(\ell)\,.$$

Next, consider a verification query (i, N_P, N_S^*, C, X), and let C be the corresponding ciphertext, Chk be the message checksum, mlen be the message length, N_S be the decrypted secret nonce, and let $T \leftarrow \mathsf{G}_H(N_S, \mathrm{Chk}, \mathsf{mlen})$ be the secret tag. For this verification query to set $\mathsf{bad} \leftarrow \mathbf{true}$, there must be a prior encryption query (i, N_P, N_S^*, M', X') with corresponding ciphertext C' and secret tag $T' \leftarrow \mathsf{G}_H(N_S^*, \mathrm{Chk}', \mathsf{mlen}')$ such that $T \oplus X = T' \oplus X'$. We proceed by case analysis.

Case 1: $(N_S, C) = (N_S^*, C')$. Since both queries use the same keys $(\tilde{K}_i, H_i)$ and the same public nonce N_P, the tags are equal $T = T'$. Then to set $\mathsf{bad} \leftarrow \mathbf{true}$, we need $X = X'$, but this would make the verification query invalid, so this never sets

$\mathsf{bad} \leftarrow \textbf{true}$.

Case 2: $(N_S, \mathsf{mlen}) \neq (N_S{}^*, \mathsf{mlen}')$. Since G is ϵ-AXU, this sets $\mathsf{bad} \leftarrow \textbf{true}$ with probability at most $\epsilon(\ell)$.

Case 3: $(N_S, \mathsf{mlen}) = (N_S{}^*, \mathsf{mlen}')$ and the first ciphertext core blocks $C_1 \neq C_1'$. Let $N_S \parallel M_* := P_1 := \pi^{-1}_{(i,N_P),1}(C_1)$. But, C_1 is not the output of a previous NaxEnc query with the same i and nonce (N_P, N_S) since nonce-respecting adversaries can only make one such query and C_1' is the output of that query. That means C_1 is a fresh query to $\pi^{-1}_{(i,N_P),1}$. Let $B_{(i,N_P)}$ be the number of previous queries to $\pi^{-1}_{(i,N_P),1}$, i.e., the number of queries to the same i and public nonce N_P. Then, P_1 assumes any of the remaining $2^w - B_{(i,N_P)}$ w-bit outputs with equal probability. Therefore,

$$\begin{aligned}\Pr[\text{Case 3 sets } \mathsf{bad}] &\leq \Pr[\mathrm{Chk} = \mathrm{Chk}'] + \Pr[T_d \oplus X = T_e \oplus X' \mid \mathrm{Chk} \neq \mathrm{Chk}'] \\ &\leq 2^{n-\kappa} \cdot \frac{2^w - B_{(i,N_P)}}{2^w} \cdot \frac{1}{2^w - B_{(i,N_P)}} + \epsilon \\ &\leq \frac{2}{2^\kappa} + \epsilon(\ell)\,,\end{aligned}$$

where we upper bound the probability of $\mathrm{Chk} = \mathrm{Chk}'$ with the following analysis. The checksum checks the first κ bits, so there are $2^{w-\kappa}$ valid w-bit strings that produce a collision. Since we are upper bounding, suppose all the previous $B_{(i,N_P)}$ queries produced invalid strings. Then all $2^{w-\kappa}$ valid strings are still left in a pool of size $2^w - B_{(i,N_P)}$. So the probability of picking a valid string is at most

$$\leq \frac{2^{w-\kappa}}{2^w - B_{(i,N_P)}} \leq \frac{2^{w-\kappa}}{2^{w-1}} \leq \frac{2}{2^\kappa}\,,$$

where the second inequality uses the assumption that $B_{(i,N_P)} \leq q < 2^{2\kappa-1} \leq 2^{w-1}$.

Case 4: $(N_S, \mathsf{mlen}) = (N_S{}^*, \mathsf{mlen}')$ and some ciphertext core blocks $C_j \neq C_j'$ for $j > 1$. As above, C_j is not the output of a previous NaxEnc query with the same i and nonce (N_P, N_S). So, the corresponding message block $P_j' := \pi^{-1}_{(i,N_P,N_S),j}(C_j')$ assumes any of the remaining $2^w - 1$ uniformly distributed n-bit outputs with equal probability. Therefore,

$$\Pr[\text{Case 4 sets } \mathsf{bad}] \leq \Pr[\text{Chk} = \text{Chk}'] + \Pr[T_d \oplus X = T_e \oplus X' \mid \text{Chk} \neq \text{Chk}'] \leq \frac{2}{2^\kappa} + \epsilon(\ell)\,.$$

Case 5: $(N_S, \mathsf{mlen}) = (N_S{}^*, \mathsf{mlen}')$ and the partial ciphertext core blocks $C_* \neq C'_*$. Then $\text{Chk} \neq \text{Chk}'$ so

$$\Pr[\text{Case 5 sets } \mathsf{bad}] = \Pr[T_d \oplus X = T_e \oplus X' \mid \text{Chk} \neq \text{Chk}'] \leq \epsilon(\ell)\,.$$

Taking the maximum of the cases, we get

$$\Pr[\text{verification query sets } \mathsf{bad}] \leq \frac{2}{2^\kappa} + \epsilon(\ell)\,.$$

Applying the union bound over at most q_v verification queries and at most q encryption queries, we get

$$\Pr[\mathrm{G}_1(\mathcal{A}) \text{ sets } \mathsf{bad}] \leq \frac{2q_v}{2^\kappa} + q_v \cdot \epsilon(\ell) + q \cdot \min(B, 2^{n_s}) \cdot \epsilon(\ell)\,.$$

Game G_2 is the random game $\mathrm{NAX}^{\mathrm{rand}}$. Since G_1's NaxEnc queries already return **false** for the win boolean and its NaxVer queries always return **false**, the gap between G_1 and G_2 is the probability of distinguishing between their NaxEnc ciphertext core outputs. Since $\mathcal{A}$ is nonce-respecting, the only time a tweak may be reused is for encrypting the first block P_1 for the same user, since that operation only uses the public nonce as the tweak. For all other TBC calls, a fresh tweak is used meaning their result is a random string. Therefore, we can use the multi-user PRP/PRF switching lemma [15, Lemma 4.1]. Summing over the maximum $\min(B, 2^{n_s})$ encryption queries to the same user i and public nonce N_P, and applying the union bound over q queries we get

$$\Pr[\mathrm{G}_2(\mathcal{A})] - \Pr[\mathrm{G}_1(\mathcal{A})] \leq \frac{q \cdot \min(B, 2^{n_s})}{2^w}\,.$$

Putting it all together, we get

$$\mathsf{Adv}^{\mathrm{nax}}_{\Theta\mathsf{CX}[\tilde{E},\mathsf{G}]}(\mathcal{A}) \leq \Pr[\mathrm{G}_0(\mathcal{A})] - \Pr[\mathrm{G}_2(\mathcal{A})] \leq \frac{q \cdot \min(B, 2^{n_s})}{2^w} + \frac{2q_v}{2^\kappa} + q_v \cdot \epsilon(\ell) + q \cdot \min(B, 2^{n_s}) \cdot \epsilon(\ell)\,.$$

□

Theorem 21. *Let* $\tilde{E} : \{0,1\}^{\tilde{k}} \times \mathcal{T}_{\Theta\mathsf{CH}} \times \{0,1\}^w \to \{0,1\}^w$ *be a* w*-bit TBC with the* $\Theta\mathsf{CH}$ *tweakspace* $\mathcal{T}_{\Theta\mathsf{CH}}$*. Let* $\mathsf{G} : \{0,1\}^h \times \{0,1\}^* \to \{0,1\}^\tau$ *be an* ϵ*-AXU hash function. Let* $\mathcal{A}$ *be an NAX adversary making at most* $q < 2^{2\kappa-1}$ *queries totalling* σ *blocks, with at most* q_v *verification queries. Assume* $\mathcal{A}$ *queries* u *users, making at most* B *queries (encryption and verification) per user totalling* s *blocks per user. Then, we can construct a TPRP adversary* $\mathcal{B}$ *such that*

$$\mathsf{Adv}^{\mathrm{nax}}_{\Theta\mathsf{CX}[\tilde{E},\mathsf{G}]}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{tprp}}_{\tilde{E}}(\mathcal{B}) + \frac{q \cdot \min(B, 2^{n_s})}{2^w} + \frac{2q_v}{2^\kappa} + q_v \cdot \epsilon(\ell) + q \cdot \min(B, 2^{n_s}) \cdot \epsilon(\ell)\,,$$

where $\ell := 8 + n_s + w + r$ *is the maximum length that* $\Theta\mathsf{CX}$ *calls the AXU hash* G *on. Adversary* $\mathcal{B}$ *makes at most* $u + \sigma$ *TPRP queries and at most* s *TPRP queries per user, and has time complexity similar to* $\mathcal{A}$*.*

Proof. Consider the following sequence of games. Game G_0 is the real game $\mathrm{NAX}^{\mathrm{real}}$, game G_1 is identical except it replaces calls to the real TBC $\tilde{E}$ with calls to an ideal TBC $\tilde{\pi} \leftarrow\!\!{\scriptstyle\$}\ \widetilde{\mathrm{Perms}}(w, \mathcal{T}_{\Theta\mathsf{CH}})$. We bound the gap between these games by constructing a TPRP adversary $\mathcal{B}$ for $\tilde{E}$ that emulates $\mathrm{G}_0(\mathcal{A})$ but runs TPRPKG for each user and replaces calls to $\tilde{E}$ and $\tilde{D}$ with calls to the TPRP game's TPRPENC and TPRPDEC oracles. Then

$$\Pr[\mathrm{G}_0(\mathcal{A})] - \Pr[\mathrm{G}_1(\mathcal{A})] \leq \mathsf{Adv}^{\mathrm{tprp}}_{\tilde{E}}(\mathcal{B})\,,$$

and $\mathcal{B}$ makes at most $u + \sigma$ TPRP queries and at most s TPRP queries per user, and has time complexity similar to $\mathcal{A}$.

Game G_2 is the random game $\mathrm{NAX}^{\mathrm{rand}}$. Since the $\mathrm{NAX}^{\mathrm{rand}}$ game does not depend on the choice of TBC; that is, it produces the same output for a real TBC and an ideal TBC, we can apply Lemma 20 to get

$$\Pr[\mathrm{G}_2(\mathcal{A})] - \Pr[\mathrm{G}_1(\mathcal{A})] = \mathsf{Adv}^{\mathrm{nax}}_{\Theta\mathsf{CX}[\tilde{\pi},\mathsf{G}]}(\mathcal{A}) \le \frac{q \cdot \min(B, 2^{n_s})}{2^w} + \frac{2q_v}{2^\kappa} + q_v \cdot \epsilon(\ell) + q \cdot \min(B, 2^{n_s}) \cdot \epsilon(\ell)\,.$$

We get the claimed bound by subtracting these differences. □

3.9.2 Proof of Theorem 13

First, let's use Theorem 8 on $\mathcal{A}$ to construct an orderly NAE adversary $\mathcal{A}'$ such that

$$\mathsf{Adv}^{\mathrm{nae}}_{\mathsf{XtH}[\mathsf{F},\mathsf{AEAD}]}(\mathcal{A}) \le \mathsf{Adv}^{\mathrm{nae}}_{\mathsf{XtH}[\mathsf{F},\mathsf{AEAD}]}(\mathcal{A}') + \frac{q}{2^{2\kappa}}\,,$$

where 2κ is the ciphertext overhead of $\Theta\mathsf{CH}$ and adversary $\mathcal{A}'$ makes at most q queries, with at most q_v verification queries, to u users and at most B queries per user, and has running time similar to $\mathcal{A}$. Without loss of generality, let's assume that the claimed bound is less than 1, and consider the following sequence of games.

Game G_0 is the real game $\mathrm{NAE}^{\mathrm{real}}$, and game G_1 is identical except it samples subkeys at random and replaces the PRF $\mathsf{F}^{2\kappa}_{K_i}(\cdot)$ with a truly random function $f_i : \{0,1\}^* \to \{0,1\}^{2\kappa}$. Then, we can bound the gap between G_0 and G_1 by constructing a PRF adversary $\mathcal{B}$ for the PRF F which emulates $\mathrm{G}_0(\mathcal{A}')$ but replaces calls to $\mathsf{F}_{K_i}(\cdot)$ with the PRF game's $\textsc{Func}(i, \cdot)$. Then

$$\Pr[\mathrm{G}_0(\mathcal{A}')] - \Pr[\mathrm{G}_1(\mathcal{A}')] \le \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{B})\,,$$

and $\mathcal{B}$ makes at most $2q$ $\textsc{Func}$ queries and has time complexity similar to $\mathcal{A}$.

Game G_2 is identical to G_1 except it replaces the TBC $\tilde{E}(\tilde{K}_i, t, \cdot)$ with a truly random permutation $\tilde{\pi}_{i,t}$. We can bound the gap between these games by constructing a TPRP adversary C for $\tilde{E}$ that emulates $\mathrm{G}_1(\mathcal{A}')$ but runs TprpKg for each user and replaces calls to $\tilde{E}$ and $\tilde{D}$ with calls to the TPRP game's TprpEnc and TprpDec oracles. Then

$$\Pr[\mathrm{G}_1(\mathcal{A}')] - \Pr[\mathrm{G}_2(\mathcal{A}')] \leq \mathsf{Adv}^{\mathrm{tprp}}_{\tilde{E}}(C) ,$$

and C makes at most $u + \sigma$ TPRP queries and at most s TPRP queries per user, and has time complexity similar to $\mathcal{A}$.

Game G_3 is identical to G_2 but for every encryption query (i, N_P, N_S, A, M) with $|N_S| + |M| < n$ we pick a truly random ciphertext, for every verification query $(i, N_P, A, C^* \parallel T^*)$ with $|C^*| < n$ we return **false**. To bound the gap between G_1 and G_2, let Bad be the event that there are two encryption queries (i, N_P, N_S, A, M) and (i, N_P, N_S', A', M') in game G_2 such that $|N_S| + |M| < n$ and $\mathsf{ixor}(A, T) = \mathsf{ixor}(A', T')$, where T and T' are the internal tags of those queries.

We now bound the chance that Bad happens. Without loss of generality, assume that we terminate the game immediately when Bad happens. Then, the AXU hash keys are independent of what $\mathcal{A}$ receives, and are independent of $\mathcal{A}$'s queries. Fix an encryption query (i, N_P, N_S, A, M) with $|N_S| + |M| < n$, and with internal tag $T \leftarrow \mathsf{G}_H(N_S \parallel M)$. For this encryption query to cause Bad, there must be another encryption query (i, N_P, N_S', A', M') with internal tag $T' \leftarrow \mathsf{G}_H(N_S', \cdot)$ such that $\mathsf{ixor}(A, T) = \mathsf{ixor}(A', T')$. Recall that $\mathsf{ixor}(A, T) = \mathsf{ixor}(A', T')$ implies $\mathsf{prefix}(A) \oplus T = \mathsf{prefix}(A') \oplus T'$. And since G is ϵ-AXU and nonce uniqueness implies $N_S \neq N_S'$, we get

$$\Pr[\text{this causes } \mathsf{Bad}] \leq \epsilon ,$$

where $\ell := 8 + n_s + w + r$ is the maximum length that $\Theta\mathsf{CX}$ calls the AXU hash G on. Summing this over the maximum $\min(B, 2^{n_s})$ encryption queries to the same user i and public nonce N_P, we get

$$\Pr[\text{encryption query causes } \mathsf{Bad}] \le \min(B, 2^{n_s}) \cdot \epsilon\,.$$

Summing over the maximum q encryption queries, we get

$$\Pr[\mathsf{Bad}] \le q \cdot \min(B, 2^{n_s}) \cdot \epsilon\,.$$

Assume that Bad does not happen, then in game G_2 the adversary receives no information about the internal tags of encryption queries with $|N_S| + |M| < n$.

We now bound the chance that there is some verification query $(i, N_P, A, C^* \parallel T^*)$ with $|C^*| < n$ that returns **true**. Since $\mathcal{A}'$ is orderly, its verification queries only depend on encryption queries. So, without loss of generality, assume that this is the first verification query. Let (M, N_S) be the tentative decrypted answer for this query and let $T \leftarrow \mathsf{G}_H(N_S \parallel M)$ be the internal tag. If $(5, N_P, \mathsf{ixor}(A, T))$ is a fresh query to f_i then the verification query returns **true** with probability at most $2^{-2\kappa}$. It remains to bound the chance that there is a prior encryption query (i, N_P, N_S', A', M') with $|N_S| + |M| < n$ and internal tag $T' \leftarrow \mathsf{G}_H(N_S' \parallel M')$ that calls $f_i(5, N_P, \mathsf{ixor}(A', T'))$ with $\mathsf{ixor}(A, T) = \mathsf{ixor}(A', T')$. We proceed by case analysis.

Case 1: $(\mathsf{prefix}(A), N_S \parallel M) \ne (\mathsf{prefix}(A'), N_S' \parallel M')$. Then $\mathsf{ixor}(A, T) = \mathsf{ixor}(A', T')$ implies $\mathsf{prefix}(A) \oplus T = \mathsf{prefix}(A') \oplus T'$. And since G is ϵ-AXU, this happens with probability at most ϵ. Summing this over the maximum $\min(B, 2^{n_s})$ encryption queries to the same user i and public nonce N_P, we get

$$\Pr[\text{Case 1 causes } \textbf{true}] \le \min(B, 2^{n_s}) \cdot \epsilon\,.$$

Case 2: $(\mathsf{prefix}(A), N_S \parallel M) = (\mathsf{prefix}(A'), N_S' \parallel M')$. Then $\mathsf{ixor}(A, T) = \mathsf{ixor}(A', T')$

implies $\mathsf{prefix}(A) \oplus T = \mathsf{prefix}(A') \oplus T'$ which implies $T = T'$. This in turn implies $A = A'$. But, the same nonce (N_P, N_S), the same associated data A, the same message M, and the same internal tag T, implies that the encryption query returns the same outer tag T^* and ciphertext core C^* which makes this verification query invalid.

Summing over the q queries, the chance that some verification query with $|C^*| < n$ returns **true** is at most

$$\Pr[\text{returns } \textbf{true}] \le \frac{q}{2^{2\kappa}} + q \cdot \min(B, 2^{n_s}) \cdot \epsilon\,.$$

Now for encryption queries (i, N_P, N_S, A, M) with $|N_S| + |M| < n$, their outer tags T^* are uniformly random 2κ-bit strings. As long as there is no collision among outer tags T^* for the same user i and public nonce N_P, each ciphertext cores C^* is generated using a fresh TBC call, producing a random string. From the multi-user PRP/PRF switching lemma [15, Lemma 4.1], the probability of distinguishing these ciphertext cores from truly random ones is at most $qB/2^{2\kappa}$.

Hence,

$$\Pr[\mathrm{G}_2(\mathcal{A}')] - \Pr[\mathrm{G}_3(\mathcal{A}')] \le \frac{q(B+1)}{2^{2\kappa}} + 2q \cdot \min(B, 2^{n_s}) \cdot \epsilon\,.$$

Game G_4 is the random game $\mathrm{NAE}^{\mathrm{rand}}$. Without loss of generality, we can assume that the adversary does not make encryption queries with $|N_S|+|M| < n$ nor verification queries with $|C^*| < n$, since those queries are identical between the two games.

Then, we bound the gap between G_3 and G_4 by constructing an NAE adversary against $\mathsf{XtH}[\tilde{F}, \Theta\mathsf{CX}[\tilde{\pi}, \mathsf{G}]]$ with a ideal PRF $\tilde{F}$ and an ideal TBC $\tilde{\pi}$. Using Lemma 19 and Lemma 20, we get

$$\begin{aligned}\Pr[\mathrm{G}_4(\mathcal{A}')] - \Pr[\mathrm{G}_3(\mathcal{A}')] &= \mathsf{Adv}^{\mathrm{nae}}_{\mathsf{XtH}[\tilde{F},\Theta\mathsf{CX}[\tilde{\pi},\mathsf{G}]]}(\mathcal{A}') \\ &\leq \mathsf{Adv}^{\mathrm{nax}}_{\Theta\mathsf{CX}[\tilde{\pi},\mathsf{G}]}(\mathcal{D}) + \frac{2qB-q}{2^{2\kappa}} \\ &\leq \frac{2q_v}{2^{\kappa}} + \frac{2qB-q}{2^{2\kappa}} + \frac{q\cdot\min(B,2^{n_s})}{2^{w}} + (q_v + \min(B,2^{n_s}))\epsilon(\ell)\,.\end{aligned}$$

Putting it all together,

$$\begin{aligned}\mathsf{Adv}^{\mathrm{nae}}_{\Theta\mathsf{CH}[\tilde{E},\mathsf{G},\mathsf{F}]}(\mathcal{A}) \leq& \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{B}) + \mathsf{Adv}^{\mathrm{tprp}}_{\tilde{E}}(\mathcal{C}) + \frac{2q_v}{2^{\kappa}} + \frac{q(3B+1)}{2^{2\kappa}} \\ &+ \frac{q\cdot\min(B,2^{n_s})}{2^{w}} + (q_v + 2q\min(B,2^{n_s}))\cdot\epsilon\,.\end{aligned}$$

□

3.10 TPRP Security of OCT

In this section, we will prove the TPRP security of OCT in the ideal permutation model, as stated in Theorem 14. But first we consider the SPRP and TPRP security of Even-Mansour and tweakable Even-Mansour, respectively.

SPRP security of Even-Mansour. First, we consider the multi-user SPRP security of the padded-key Even-Mansour blockcipher (defined in Figure 3.22) in the ideal permutation model. This was shown by ADMA [7, Theorem 2 & 3] (see also [132, Theorem 1]) and we give a simplified version of their result below in Theorem 22.

Theorem 22 ([7]). *Fix a block size $w \geq 2\kappa$, and let $\Pi \leftarrow_{\$} \mathrm{Perm}(\{0,1\}^w)$ be a w-bit ideal permutation. Let $\mathcal{A}$ be an SPRP adversary against* EM *making q SPRP queries (across all users) and p ideal permutation queries. Then,*

$$\mathsf{Adv}^{\mathrm{sprp}}_{\mathsf{EM}}(\mathcal{A}) \leq \frac{q^2+2qp}{2^{2\kappa}}\,.$$

```
$\underline{\mathsf{EM}^{\Pi^\pm}.\mathsf{Enc}(K, X)}$:
$\triangleright \{0,1\}^{2\kappa} \times \{0,1\}^w \to \{0,1\}^w$
$\Delta_K \leftarrow K \,\|\, 0^{w-2\kappa}$
return $\Pi(\Delta_K \oplus X) \oplus \Delta_K$

$\underline{\mathsf{EM}^{\Pi^\pm}.\mathsf{Dec}(K, Y)}$:
$\triangleright \{0,1\}^{2\kappa} \times \{0,1\}^w \to \{0,1\}^w$
$\Delta_K \leftarrow K \,\|\, 0^{w-2\kappa}$
return $\Pi^{-1}(\Delta_K \oplus Y) \oplus \Delta_K$

$\underline{\mathsf{TEM}^{\Pi^\pm}[\mathsf{G}].\mathsf{Enc}(K, T, X)}$:
$\triangleright \{0,1\}^{2\kappa} \times \mathcal{T} \times \{0,1\}^w \to \{0,1\}^w$
$\Delta_{K,T} \leftarrow K \,\|\, 0^{w-2\kappa}$
return $\Pi(\Delta_{K,T} \oplus X) \oplus \Delta_{K,T}$

$\underline{\mathsf{TEM}^{\Pi^\pm}[\mathsf{G}].\mathsf{Dec}(K, T, Y)}$:
$\triangleright \{0,1\}^{2\kappa} \times \mathcal{T} \times \{0,1\}^w \to \{0,1\}^w$
$\Delta_{K,T} \leftarrow \mathsf{G}(K, T)$
return $\Pi^{-1}(\Delta_{K,T} \oplus Y) \oplus \Delta_{K,T}$
```

```
$\underline{\mathsf{OH}^{\Pi^\pm}(K, T)}$:
$\triangleright \{0,1\}^{2\kappa} \times \mathcal{T}_{\Theta\mathsf{CH}} \to \{0,1\}^{2\kappa}$
$L \leftarrow \mathsf{EM}^{\Pi^\pm}.\mathsf{Enc}(K, 0^w)[1 : 2\kappa]$
match $T$                              // structural pattern matching
case $(N_P, j) \leftarrow T$ then
    $R_{N_P} \leftarrow \mathsf{EM}^{\Pi^\pm}.\mathsf{Enc}(K, N_P \,\|\, 0^{w-n_p-2} \,\|\, 10)[1 : 2\kappa]$
    return $(((j+2) \cdot L) \oplus R_{N_P})$
case $(N_P, N_S, i, j) \leftarrow T$ then
    $\mathsf{Top} \leftarrow N_P \,\|\, N_S[1 : n_s - 6]$
    $\mathsf{KTop} \leftarrow \mathsf{EM}^{\Pi^\pm}.\mathsf{Enc}(K, \mathsf{Top} \,\|\, 0^{w-n_p-n_s+4} \,\|\, 01)$
    $\mathsf{Bottom} \leftarrow \mathsf{str2int}(N_S[n_s - 5 : n_s])$
    $R_{N_P,N_S} \leftarrow \mathsf{SH}(\mathsf{KTop}, \mathsf{Bottom})$
    return $(((4\gamma_i + j) \cdot L) \oplus R_{N_P,N_S})$

$\underline{\mathsf{SH}(\mathsf{KTop}, \mathsf{Bottom})}$:
$\triangleright \{0,1\}^w \times [1..63] \to \{0,1\}^{2\kappa}$
if $w < (2\kappa + 64)$ then
    $\mathsf{TKTop} \leftarrow \mathsf{KTop}[1 : 2\kappa]$
    $\mathsf{KTopStretch} \leftarrow \mathsf{TKTop} \| (\mathsf{TKTop} \oplus (\mathsf{TKTop} \ll c))$
elseif $w \geq (2\kappa + 64)$ then
    $\mathsf{KTopStretch} \leftarrow \mathsf{KTop}$
return $(\mathsf{KTopStretch} \ll \mathsf{Bottom})[1 : 2\kappa]$
```

```
$\underline{\mathsf{OCT}^{\Pi^\pm}[\mathsf{G}].\mathsf{Enc}(K, T, X)}$:
$\triangleright \{0,1\}^{2\kappa} \times \mathcal{T}_{\Theta\mathsf{CH}} \times \{0,1\}^w \to \{0,1\}^w$
return $\mathsf{TEM}^{\Pi^\pm}[\mathsf{OH}^{\Pi^\pm}].\mathsf{Enc}(K, T, X)$

$\underline{\mathsf{OCT}^{\Pi^\pm}[\mathsf{G}].\mathsf{Dec}(K, T, Y)}$:
$\triangleright \{0,1\}^{2\kappa} \times \mathcal{T}_{\Theta\mathsf{CH}} \times \{0,1\}^w \to \{0,1\}^w$
return $\mathsf{TEM}^{\Pi^\pm}[\mathsf{OH}^{\Pi^\pm}].\mathsf{Dec}(K, T, Y)$
```

Figure 3.22: Modular definitions of (Left) the Even-Mansour blockcipher EM and the tweakable Even-Mansour TBC TEM, (Middle) the offset function OH, and (Right) the tweakable blockcipher OCT. All constructions are based on a permutation $\Pi : \{0,1\}^w \to \{0,1\}^w$ with blocksize $w \geq 2\kappa$. The choice of constant c in SH is discussed in Lemma 24.

TPRP security of tweakable Even-Mansour. Next, we consider the multi-user TPRP security of tweakable Even-Mansour TBC $\mathsf{TEM}[\mathsf{G}]$ (defined in Figure 3.22) in the ideal permutation model, with an the offset function G that is *independent* of the permutation Π. This was previously considered by Zhang and Hu [168], but unfortunately their proof is flawed because of an incorrect combination of inequalities. Still, the proof can be easily fixed, and the updated bound is given in Theorem 23 below with the corrected proof is in Appendix 3.11.

Theorem 23. *Fix a block size $w \geq 2\kappa$. Let $\Pi \leftarrow_\$ \mathrm{Perm}(\{0,1\}^w)$ be a w-bit ideal permutation, let $\mathcal{T}$ be the tweakspace, let $H : \{0,1\}^{2\kappa} \times \mathcal{T} \to \{0,1\}^w$ be an ϵ-AXU and δ-uniform offset function (that is independent of the ideal permutation Π). Let $\mathcal{A}$ be a TPRP adversary making q TPRP queries and p ideal permutation queries. Assume $\mathcal{A}$ queries u users,*

κ	c																				
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	32
64	3	15	7	3	124	7	3	**85**	120	3	118	63	3	31	63	3	7	31	3	7	15
128	255	3	31	15	3	7	250	3	248	247	3	7	7	3	242	**85**	3	239	7	3	3
256	3	511	7	3	7	31	3	15	15	3	63	7	3	499	498	3	7	341	3	341	**85**

Figure 3.23: maxBottom for various choices of constant c. The selected values of c are highlighted.

making at most B TPRP queries (encryption and verification) per user. Then,

$$\mathsf{Adv}^{\mathrm{tprp}}_{\mathsf{TEM}}(\mathcal{A}) \leq qB \cdot \epsilon + q(q+2p) \cdot \delta \,.$$

AXU security of the SH hash. Now, we consider the δ-uniform and ϵ-AXU security of the hash $\mathsf{SH} : \{0,1\}^w \times [1..63] \to \{0,1\}^{2\kappa}$ (defined in Figure 3.22). SH generalizes the Stretch-then-Shift hash in OCB3 [112, §7.2], and our analysis is an extension of the analysis in OCB3 [112, §7.2]. We state the result below in Lemma 24.

Lemma 24. *Fix a security parameter $\kappa \in \{64, 128, 256\}$ and a block size $w \geq 2\kappa$. Then $\mathsf{SH} : \{0,1\}^w \times [1..63] \to \{0,1\}^{2\kappa}$ (defined in Figure 3.22) is δ-uniform and ϵ-AXU, with $\delta = \epsilon = 2^{-2\kappa}$. That is, for any input $\mathsf{Bottom} \in [1..63]$, any distinct inputs $\mathsf{Bottom}_1, \mathsf{Bottom}_2 \in [1..63]$, and any output $\Delta \in \{0,1\}^{2\kappa}$, it holds that*

$$\Pr_{\mathsf{KTop} \twoheadleftarrow\$ \{0,1\}^w}[\mathsf{SH}(\mathsf{KTop}, \mathsf{Bottom}) = \Delta] \leq 2^{-2\kappa}\,,$$

$$\Pr_{\mathsf{KTop} \twoheadleftarrow\$ \{0,1\}^w}[\mathsf{SH}(\mathsf{KTop}, \mathsf{Bottom}_1) \oplus \mathsf{SH}(\mathsf{KTop}, \mathsf{Bottom}_2) = \Delta] \leq 2^{-2\kappa}\,.$$

Proof. Let's first consider the simpler case when blocksize $w \geq 2\kappa + 64$. Then $\mathsf{SH}(\mathsf{KTop}, \mathsf{Bottom}) = (\mathsf{KTop} \ll \mathsf{Bottom})[1 : 2\kappa]$ selects 2κ bits from the first $2\kappa + 64$ bits of KTop. So for any input $\mathsf{Bottom} \in [1..63]$ and any output $\Delta \in \{0,1\}^{2\kappa}$, it holds that

$$\Pr_{\mathsf{KTop} \twoheadleftarrow\$ \{0,1\}^w}[\mathsf{SH}(\mathsf{KTop}, \mathsf{Bottom}) = \Delta \mid w \geq 2\kappa + 64] \leq 2^{-2\kappa}\,.$$

Now, fix some distinct inputs $\mathsf{Bottom}_1 < \mathsf{Bottom}_2$ and some output Δ. Then the statement $\mathsf{SH}(\mathsf{KTop}, \mathsf{Bottom}_1) \oplus \mathsf{SH}(\mathsf{KTop}, \mathsf{Bottom}_2)$ implies that $\mathsf{KTop}[i] \oplus \mathsf{KTop}[i + (\mathsf{Bottom}_2 - \mathsf{Bottom}_1)]$ for every $i \in \{\mathsf{Bottom}_1 + 1, \dots, \mathsf{Bottom}_1 + 2\kappa\}$. This further implies that $\mathsf{KTop}[\mathsf{Bottom}_2 + 1 : \mathsf{Bottom}_2 + 2\kappa]$ is uniquely determined from $\mathsf{KTop}[\mathsf{Bottom}_1 + 1 : \mathsf{Bottom}_2]$ and Δ, which over the choice of KTop happens with probability at most $2^{-2\kappa}$. Thus,

$$\Pr_{\mathsf{KTop} \leftarrow\$ \{0,1\}^w}[\mathsf{SH}(\mathsf{KTop}, \mathsf{Bottom}_1) \oplus \mathsf{SH}(\mathsf{KTop}, \mathsf{Bottom}_2) = \Delta \mid w \geq 2\kappa + 64] \leq 2^{-2\kappa}\,.$$

Let's now analyze the more complex case when blocksize $w \in \{2\kappa, \dots, 2\kappa + 64\}$ with security parameter $\kappa \in \{64, 128, 256\}$. Our analysis follows that of the Stretch-then-Shift hash in OCB3 [112, §7.2]. For any fixed value of the constant c, following OCB3 [112, §7.2], we construct $2\kappa \times 2\kappa$ matrices A_i for i such that $\mathsf{SH}(\mathsf{KTop}, i) = A_i \mathsf{TKTop}$, where $\mathsf{TKTop} = \mathsf{KTop}[1 : 2\kappa]$ is interpreted as a column vector of size $|\mathsf{KTop}| \in \{128, 256, 512\}$ and multiplication is in GF(2). Then, $2^{-2\kappa}$-uniformity is equivalent to the matrices A_i being invertible for all i, and $2^{-2\kappa}$-AXU is equivalent to the matrix sums $(A_i + A_j)$ being invertible for all distinct i, j. Following this, we wrote a SageMath [154] program to compute the largest value $\mathsf{maxBottom}$ where A_i is invertible for all $i < \{0, \dots, \mathsf{maxBottom} - 1\}$ and $(A_i + A_j)$ is invertible for all distinct $i, j \in \{0, \dots, \mathsf{maxBottom} - 1\}$. We show our results for various choices of c in Figure 3.23. Thankfully, for each $\kappa \in \{64, 128, 256\}$, there are many small choices of c with $\mathsf{maxBottom} \geq 64$. For simplicity, like OCB3 [112, §7.2], we select multiples of eight; that is $(\kappa, c) \in \{(64, 8), (128, 16), (256, 32)\}$. □

Proof of Theorem 14. We will transition from $\mathrm{TPRP}^{\mathrm{real}}$ to $\mathrm{TPRP}^{\mathrm{rand}}$ using a sequence of games defined in Figure 3.25.

```
$\textsc{Off}^{\Gamma_i}(T)$:
$\triangleright\ \mathcal{T}_{\Theta\mathsf{CH}} \to \{0,1\}^{2\kappa}$
$L \leftarrow \Gamma_i(0^w)[1 : 2\kappa]$            // cached for each user
match $T$                     // structural pattern matching
case $(N_P, j) \leftarrow T$ then
    $R_{N_P} \leftarrow \Gamma_i(N_P \,\|\, 0^{w-n_p-2} \,\|\, 10)[1 : 2\kappa]$
    return $(((j+2)\cdot L) \oplus R_{N_P})$
case $(N_P, N_S, i, j) \leftarrow T$ then
    $\mathsf{Top} \leftarrow N_P \,\|\, N_S[1 : n_s - 6]$
    $\mathsf{KTop} \leftarrow \Gamma_i(\mathsf{Top} \,\|\, 0^{w-n_p-n_s+4} \,\|\, 01)$
    $\mathsf{Bottom} \leftarrow \mathsf{str2int}(N_S[n_s - 5 : n_s])$
    $R_{N_P,N_S} \leftarrow \mathsf{SH}(\mathsf{KTop}, \mathsf{Bottom})$        // def in Fig 3.22
    return $(((4\gamma_i + j)\cdot L) \oplus R_{N_P,N_S})$
endmatch
```

Figure 3.24: Procedure Off used in proof of Theorem 14. It calls subprocedure SH defined in Figure 3.22.

```
Game $G_0(\mathcal{A})$:
$\Pi \leftarrow\!\!\$\ \mathrm{Perm}(\mathcal{M})$
$b \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Tprp}\{\textsc{Kg},\textsc{Enc},\textsc{Dec}\},\Pi^\pm}$
return $b$

$\textsc{TprpKg}()$:
$u \leftarrow u + 1$
$K_u \leftarrow\!\!\$\ \mathcal{K}$

$\textsc{TprpEnc}(i, T, X)$:
$\Delta_{i,T} \leftarrow \mathsf{OH}^{\Pi^\pm}(K_i, T)$
return $\Pi(\Delta_{i,T} \oplus X) \oplus \Delta_{i,T}$

$\textsc{TprpDec}(i, T, Y)$:
$\Delta_{i,T} \leftarrow \mathsf{OH}^{\Pi^\pm}(K_i, T)$
return $\Pi^{-1}(\Delta_{i,T} \oplus Y) \oplus \Delta_{i,T}$
```

```
Game $G_2(\mathcal{A})$:
$\Pi \leftarrow\!\!\$\ \mathrm{Perm}(\mathcal{M})$
$b \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Tprp}\{\textsc{Kg},\textsc{Enc},\textsc{Dec}\},\Pi^\pm}$
return $b$

$\textsc{TprpKg}()$:
$u \leftarrow u + 1$
$\tilde{\pi}_u \leftarrow\!\!\$\ \widetilde{\mathrm{Perm}}(\mathcal{T}, \mathcal{M})$

$\textsc{TprpEnc}(i, T, X)$:
return $\tilde{\pi}_i(T, X)$

$\textsc{TprpDec}(i, T, Y)$:
return $\tilde{\pi}_i^{-1}(T, Y)$
```

```
Game $G_1(\mathcal{A})$:
$\Pi \leftarrow\!\!\$\ \mathrm{Perm}(\mathcal{M})$
$b \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Tprp}\{\textsc{Kg},\textsc{Enc},\textsc{Dec}\},\Pi^\pm}$
return $b$

$\textsc{TprpKg}()$:
$u \leftarrow u + 1$
$\Gamma_u \leftarrow\!\!\$\ \mathrm{Perm}(\{0,1\}^w)$

$\textsc{TprpEnc}(i, T, X)$:
$\Delta_{i,T} \leftarrow \textsc{Off}^{\Gamma_i}(T)$
return $\Pi(\Delta_{i,T} \oplus X) \oplus \Delta_{i,T}$

$\textsc{TprpDec}(i, T, Y)$:
$\Delta_{i,T} \leftarrow \textsc{Off}^{\Gamma_i}(T)$
return $\Pi^{-1}(\Delta_{i,T} \oplus Y) \oplus \Delta_{i,T}$
```

Figure 3.25: Games G_0-G_2 used in proof of Theorem 14. G_1 calls procedure Off defined in Figure 3.24

Game G_0 is the real game $\mathrm{TPRP}^{\mathrm{real}}$, and game G_1 identical except two changes: (1) for each user i, it samples a uniformly random permutation $\Gamma_i \leftarrow\!\!\$\ \mathrm{Perm}(\{0,1\}^w)$, and (2) it replaces each call to the real offset function $\mathsf{OH}^{\Pi^\pm}(K_i, T)$ with a call to the procedure $\textsc{Off}^{\Gamma_i}(T)$ (defined in Figure 3.25 and described next). To construct $\textsc{Off}^{\Gamma_i}(T)$, we leverage the fact that $\mathsf{OH}^{\Pi^\pm}(K_i, T)$ (defined in Figure 3.22) only uses its key K_i input and permutation $\Pi^\pm$ oracle to invoke $\mathsf{EM}^{\Pi^\pm}.\mathsf{Enc}(K_i, \cdot)$. It makes three such calls, once to compute the subkey L, once to compute R_{N_p}, and once to compute KTop. Now, in $\textsc{Off}^{\Gamma_i}(T)$ we replace these calls to $\mathsf{EM}^{\Pi^{\pm}}.\mathsf{Enc}(K_i, \cdot)$ with calls to the uniformly random permutation Γ_i. After this replacement, $\textsc{Off}^{\Gamma_i}(T)$ no longer depends on the key K_i or the permutation Π, so we can safely shed them. We bound the gap between games G_1 and G_0 by constructing a PRP adversary $\mathcal{B}$ for EM in the ideal permutation model. $\mathcal{B}$ emulates $G_1(\mathcal{A})$ but runs $\textsc{PrpKg}$ for each user i and replaces calls to the permutation $\Gamma_i(X)$ with calls to the PRP game's $\textsc{PrpEnc}(i, X)$. $\mathcal{B}$ also forwards all ideal permutation queries of $G_1(\mathcal{A})$ to its ideal permutation oracle. Then applying Theorem 22 we get

$$\Pr[G_0(\mathcal{A})] - \Pr[G_1(\mathcal{A})] \le \mathsf{Adv}^{\mathrm{prp}}_{\mathsf{EM}}(\mathcal{B}) \le \frac{q^2 + 2qp}{2^{2\kappa}} .$$

Adversary $\mathcal{B}$ queries $\textsc{PrpKg}$ once for each user, $\textsc{PrpEnc}$ once for each user to compute the subkey L in $\textsc{Off}$ (which it caches for future calls by the same user), $\textsc{PrpEnc}$ once for each $\textsc{TprpEnc}$ and $\textsc{TprpDec}$ call inside $\textsc{Off}$ to compute either R_{N_p} or KTop. In sum, $\mathcal{B}$ makes at most $q + 2u$ PRP queries, forwards the p ideal permutation queries, and has time complexity similar to $\mathcal{A}$.

Game G_2 is the random game $\mathrm{TPRP}^{\mathrm{rand}}$, and we want to bound the gap between G_1 and G_2 by constructing a TPRP adversary $\mathcal{C}$ for $\mathsf{TEM}[\textsc{Off}]$ in the ideal permutation model. Since $\textsc{Off}$ does not depend on the ideal permutation, we are in the setting of Theorem 23. Adversary $\mathcal{C}$ emulates $G_1(\mathcal{A})$ except it forwards calls to $\textsc{TprpKg}$, $\textsc{TprpEnc}$, and $\textsc{TprpDec}$ to its TPRP oracles, and forwards all ideal permutation queries to its to ideal permutation oracle. Suppose $\textsc{Off}$ is ϵ-AXU and δ-uniform for some ϵ and δ to be determined later. Then, applying Theorem 23, we get

$$\Pr[G_1(\mathcal{A})] - \Pr[G_2(\mathcal{A})] \le \mathsf{Adv}^{\mathrm{tprp}}_{\mathsf{TEM}[\textsc{Off}]}(\mathcal{C}) \le qB \cdot \epsilon + q(q + 2p) \cdot \delta .$$

Adversary $\mathcal{C}$ has the same query complexity as $\mathcal{A}$ and similar running time.

At last, it remains to compute the values ϵ and δ such that $\textsc{Off}$ is δ-uniform and ϵ-AXU. That is, for any input $T \in \mathcal{T}_{\Theta\mathsf{CH}}$, any distinct inputs $T_1, T_2 \in \mathcal{T}_{\Theta\mathsf{CH}}$, and any output $\Delta \in \{0,1\}^{2\kappa}$, it should hold that

$$\Pr_{\Gamma \leftarrow\$ \mathrm{Perm}(\{0,1\}^w)}[\textsc{Off}^{\Gamma}(T) = \Delta] \le \delta\,,$$

$$\Pr_{\Gamma \leftarrow\$ \mathrm{Perm}(\{0,1\}^w)}[\textsc{Off}^{\Gamma}(T_1) \oplus \textsc{Off}^{\Gamma}(T_1) = \Delta] \le \epsilon\,.$$

Let's start with δ-uniformity. Fix a tweak $T \in \mathcal{T}_{\Theta\mathsf{CH}}$ and output $\Delta \in \{0,1\}^{2\kappa}$, and pick $\pi \leftarrow_{\$} \mathrm{Perm}(\{0,1\}^w)$. Let $L = \pi(0^w)[1 : 2\kappa]$. First let's consider the case when $(N_P, j) \leftarrow T$. Then, there's a constant s such that $\textsc{Off}^{\Gamma}(T) = ((s \cdot L) \oplus R_{N_P})$ where $R_{N_P} = \pi(N_P \parallel 0^{w-n_p-2} \parallel 10)[1 : 2\kappa]$. Given $\pi(0^w)$, the value $\pi(N_P \parallel 0^{w-n_p-2} \parallel 10)$ is uniformly distributed over a set of $2^w - 1$ elements. Of these there are at most $2^{w-2\kappa}$ choices that lead to $R_{N_P} = (s \cdot L) \oplus \Delta$. Thus, in this case,

$$\Pr_{\Gamma \leftarrow\$ \mathrm{Perm}(\{0,1\}^w)}[\textsc{Off}^{\Gamma}(T) = \Delta \mid (N_P, j) \leftarrow T] \le \frac{2^{w-2\kappa}}{2^w - 1} = \frac{1 + (2^w-1)^{-1}}{2^{2\kappa}} \le \frac{1.5}{2^{2\kappa}}$$

Now, let's consider the case when $(N_P, N_S, i, j) \leftarrow T$. Then, there's a constant s such that $\textsc{Off}^{\Gamma}(T) = ((s \cdot L) \oplus R_{N_P,N_S})$ where $R_{N_P,N_S} = \mathsf{SH}(\mathsf{KTop}, \mathsf{Bottom})$, $\mathsf{KTop} = \pi(\mathsf{Top} \parallel 0^{w-n_p-n_s+4} \parallel 01)$, $\mathsf{Top} = N_P \parallel N_S[1 : n_s - 6]$, and $\mathsf{Bottom} = \mathsf{str2int}(N_S[n_s - 5 : n_s])$. Given $\pi(0^w)$, the random variable KTop is uniformly distributed over a set of $2^w - 1$ elements. Using the fact from Lemma 24 that SH is $2^{-2\kappa}$-uniform, there are at most $2^{w-2\kappa}$ choices of KTop that result in $\mathsf{SH}(\mathsf{KTop}, \mathsf{Bottom}) = (\Delta \oplus (s \cdot L))$. Thus, in this case too,

$$\Pr_{\Gamma \leftarrow\$ \mathrm{Perm}(\{0,1\}^w)}[\textsc{Off}^{\Gamma}(T) = \Delta \mid (N_P, N_S, i, j) \leftarrow T] \le \frac{2^{w-2\kappa}}{2^w - 1} = \frac{1 + (2^w-1)^{-1}}{2^{2\kappa}} \le \frac{1.5}{2^{2\kappa}}$$

Now, let's consider ϵ-AXU. Fix distinct tweaks $T_1, T_2 \in \mathcal{T}_{\Theta\mathsf{CH}}$ and output $\Delta \in \{0,1\}^{2\kappa}$, and pick $\pi \leftarrow_{\$} \mathrm{Perm}(\{0,1\}^w)$. Let $L = \pi(0^w)[1 : 2\kappa]$, and we proceed by case analysis.

Case 1: $(N_{P1}, j_1) \leftarrow T_1$ and $(N_{P2}, j_2) \leftarrow T_2$. Then, there's a constant s such that $\textsc{Off}^{\Gamma}(T_1) \oplus \textsc{Off}^{\Gamma}(T_2) = ((s \cdot L) \oplus R_{N_{P_1}} \oplus R_{N_{P_2}})$ where $R_{N_{P_1}} = \pi(N_{P1} \,\|\, 0^* \,\|\, 10)[1 : 2\kappa]$ and $R_{N_{P_2}} = \pi(N_{P2} \,\|\, 0^* \,\|\, 10)[1 : 2\kappa]$. Given $\pi(0^w)$ and $\pi(N_{P1} \,\|\, 0^* \,\|\, 10)$, the value $\pi(N_{P2} \,\|\, 0^* \,\|\, 10)$ is uniformly distributed over a set of $2^w - 2$ elements. Of these there are at most $2^{w-2\kappa}$ choices that lead to $R_{N_{P_2}} = (\Delta \oplus (s \cdot L) \oplus R_{N_{P_1}})$ Thus, in this case,

$$\Pr_{\Gamma \leftarrow\$ \,\mathrm{Perm}(\{0,1\}^w)}[\textsc{Off}^{\Gamma}(T_1) \oplus \textsc{Off}^{\Gamma}(T_2) = \Delta \mid \text{Case 1}] \le \frac{2^{w-2\kappa}}{2^w - 2} = \frac{1 + (2^w - 2)^{-1}}{2^{2\kappa}} \le \frac{1.5}{2^{2\kappa}}$$

Case 2: $(N_{P1}, j_1) \leftarrow T_1$ and $(N_{P2}, N_{S2}, i_2, j_2) \leftarrow T_2$. Then, there's a constant s such that $\textsc{Off}^{\Gamma}(T_1) \oplus \textsc{Off}^{\Gamma}(T_2) = ((s \cdot L) \oplus R_{N_{P_1}} \oplus R_{N_{P_2}, N_{S2}})$ where $R_{N_{P_1}} = \pi(N_{P1} \| 0^* \| 10)[1 : 2\kappa]$, $R_{N_{P_2}, N_{S2}} = \mathsf{SH}(\mathsf{KTop}_2, \mathsf{Bottom}_2)$, $\mathsf{KTop}_2 = \pi(\mathsf{Top}_2 \,\|\, 0^* \,\|\, 01)$, $\mathsf{Top}_2 = N_{P2} \,\|\, N_{S2}[1 : n_s - 6]$, and $\mathsf{Bottom}_2 = \mathsf{str2int}(N_{S2}[n_s - 5 : n_s])$. Given $\pi(0^w)$ and $\pi(N_{P1} \,\|\, 0^* \,\|\, 10)$, the random variable KTop is uniformly distributed over a set of $2^w - 2$ elements. Since we know that SH is $2^{-2\kappa}$-uniform from Lemma 24, there are at most $2^{w-2\kappa}$ choices of KTop that result in $\mathsf{SH}(\mathsf{KTop}, \mathsf{Bottom}) = (\Delta \oplus (s \cdot L) \oplus R_{N_{P_1}})$. Thus, in this case,

$$\Pr_{\Gamma \leftarrow\$ \,\mathrm{Perm}(\{0,1\}^w)}[\textsc{Off}^{\Gamma}(T_1) \oplus \textsc{Off}^{\Gamma}(T_2) = \Delta \mid \text{Case 2}] \le \frac{2^{w-2\kappa}}{2^w - 2} = \frac{1 + (2^w - 2)^{-1}}{2^{2\kappa}} \le \frac{1.5}{2^{2\kappa}}$$

Case 3: $(N_{P1}, N_{S1}, i_1, j_1) \leftarrow T_1$ and $(N_{P2}, N_{S2}, i_2, j_2) \leftarrow T_2$. Then, there's a constant s such that $\textsc{Off}^{\Gamma}(T_1) \oplus \textsc{Off}^{\Gamma}(T_2) = ((s \cdot L) \oplus R_{N_{P1}, N_{S1}} \oplus R_{N_{P2}, N_{S2}})$ where $R_{N_{P1}, N_{S1}} = \mathsf{SH}(\mathsf{KTop}_1, \mathsf{Bottom}_1)$, $\mathsf{KTop}_1 = \pi(\mathsf{Top}_1 \,\|\, 0^* \,\|\, 01)$, $R_{N_{P2}, N_{S2}} = \mathsf{SH}(\mathsf{KTop}_2, \mathsf{Bottom}_2)$, and $\mathsf{KTop}_2 = \pi(\mathsf{Top}_2 \,\|\, 0^* \,\|\, 01)$. Let's proceed by subcase analysis.

Case 3.1: $\mathsf{Top}_1 \neq \mathsf{Top}_2$ where $\mathsf{Top}_1 = N_{P1} \| N_{S1}[1 : n_s - 6]$ and $\mathsf{Top}_2 = N_{P2} \| N_{S2}[1 : n_s - 6]$. Given $\pi(0^w)$ and KTop_1, the random variable KTop_2 is uniformly distributed over a set of $2^w - 2$ elements. Since we know that SH is $2^{-2\kappa}$-uniform from Lemma 24, there are at most $2^{w-2\kappa}$ choices of KTop_2 that result in $\mathsf{SH}(\mathsf{KTop}_2, \mathsf{Bottom}_2) = (\Delta \oplus (s \cdot L) \oplus R_{N_{P1}, N_{S1}})$.

Thus, in this case,

$$\Pr_{\Gamma \leftarrow\$ \text{Perm}(\{0,1\}^w)}[\textsc{Off}^{\Gamma}(T_1) \oplus \textsc{Off}^{\Gamma}(T_2) = \Delta \mid \text{Case 3.1}] \leq \frac{2^{w-2\kappa}}{2^w - 2} = \frac{1 + (2^w - 2)^{-1}}{2^{2\kappa}} \leq \frac{1.5}{2^{2\kappa}}$$

Case 3.2: $\mathsf{Top}_1 = \mathsf{Top}_2$ but $\mathsf{Bottom}_1 \neq \mathsf{Bottom}_2$ where $\mathsf{Bottom}_1 = \mathsf{str2int}(N_{S1}[n_s - 5 : n_s])$, and $\mathsf{Bottom}_2 = \mathsf{str2int}(N_{S2}[n_s - 5 : n_s])$. Given $\pi(0^w)$, the random variable $\mathsf{KTop} := \mathsf{KTop}_1 = \mathsf{KTop}_2$ is uniformly distributed over a set of $2^w - 1$ elements. Since SH is $2^{-2\kappa}$-AXU from Lemma 24, we have that there are at most $2^{w-2\kappa}$ choices of KTop that result in $\mathsf{SH}(\mathsf{KTop}, \mathsf{Bottom}_1) \oplus \mathsf{SH}(\mathsf{KTop}, \mathsf{Bottom}_2) = (\Delta \oplus (s \cdot L))$. Thus, in this case,

$$\Pr_{\Gamma \leftarrow\$ \text{Perm}(\{0,1\}^w)}[\textsc{Off}^{\Gamma}(T_1) \oplus \textsc{Off}^{\Gamma}(T_2) = \Delta \mid \text{Case 3.2}] \leq \frac{2^{w-2\kappa}}{2^w - 1} = \frac{1 + (2^w - 1)^{-1}}{2^{2\kappa}} \leq \frac{1.5}{2^{2\kappa}}$$

Case 3.3: $\mathsf{Top}_1 = \mathsf{Top}_2$ and $\mathsf{Bottom}_1 \neq \mathsf{Bottom}_2$, but $(i_1, j_1) \neq (i_2, j_2)$. Then $\textsc{Off}^{\Gamma}(T_1) \oplus \textsc{Off}^{\Gamma}(T_2) = (s \cdot L)$ where the constant $s = (4\gamma_{i_1} + j_1) \oplus (4\gamma_{i_2} + j_2)$. Since the 2κ-bit Gray code γ is a permutation on $\mathbb{Z}_{2^n}$, $i_1, i_2 \leq 2^{\kappa-3}$ and $j_1, j_2 \in \{0, 1, 2, 3\}$, it follows that the constant s is non-zero. Thus, in this case,

$$\Pr_{\Gamma \leftarrow\$ \text{Perm}(\{0,1\}^w)}[\textsc{Off}^{\Gamma}(T_1) \oplus \textsc{Off}^{\Gamma}(T_2) = \Delta \mid \text{Case 3.3}] \leq \frac{2^{w-2\kappa}}{2^w} = \frac{1}{2^{2\kappa}} .$$

Maximizing over all cases, we get $\textsc{Off}$ is δ-uniform and ϵ-AXU with $\delta = \epsilon = 1.5 \cdot 2^{-2\kappa}$. Subtracting differences completes the proof. □

3.11 TPRP security of tweakable Even-Mansour

In this section, we will prove the TPRP security of $\mathsf{TEM}[\mathsf{G}]$ in the ideal permutation model, as stated in Theorem 23. The proof uses Patarin's H-coefficient technique [138, 54], which we first recall.

The H-coefficient Technique. We start with systems and transcripts. Following the notation from [96], (which was in turn inspired by Maurer's framework [123]), it is convenient to consider interactions of a distinguisher $\mathcal{A}$ with an abstract system S which answers $\mathcal{A}$'s queries. The resulting interaction then generates a transcript $\theta = \big((X_1, Y_1), \ldots, (X_q, Y_q)\big)$ of query-answer pairs. It is known that S is entirely described by the probabilities $\mathsf{p}_{\mathrm{S}}(\theta)$ that correspond to the system S responding with answers as indicated by θ when the queries in θ are made.

Now, the H-coefficient technique considers a deterministic distinguisher $\mathcal{A}$ that tries to distinguish a "real" system S_1 from an "ideal" system S_0. The adversary's interactions with those systems define transcripts $\mathcal{T}_1$ and $\mathcal{T}_0$, respectively. The following result bounds the distinguishing advantage of $\mathcal{A}$.

Lemma 25 ([138, 54])**.** *Suppose we can partition the set of valid transcripts for the ideal system into good and bad ones. Further, suppose that there exists a constant $\epsilon \geq 0$ such that $1 - \frac{\mathsf{p}_{\mathrm{S}_1}(\theta)}{\mathsf{p}_{\mathrm{S}_0}(\theta)} \leq \epsilon$ for every good transcript θ. Then, the advantage of $\mathcal{A}$ in distinguishing S_1 and S_0 is at most $\epsilon + \Pr[\mathcal{T}_0$ is bad$]$.*

Proof of Theorem 23. Since we consider computationally unbounded adversaries, without loss of generality, assume that $\mathcal{A}$ is deterministic. Assume further that $\mathcal{A}$ does not make redundant queries. That is, (1) it does not repeat prior queries, and (2) if it queries $Y \leftarrow \Pi(X)$ then it will not query $\Pi^{-1}(Y)$ and vice versa, and (3) if it queries

$C \leftarrow \textsc{TprpEnc}(i, T, M)$ then it will not query $\textsc{TprpDec}(i, T, C)$ and vice versa. When the adversary finishes querying, we will grant it all the keys. (If we are in the random world then we sample fresh keys and grant them to the adversaries.) This key revealing can only help the adversary.

Defining bad transcripts. A transcript consists of the revealed keys K_i and the following information:

- For each query $Y \leftarrow \Pi(X)$, we store a corresponding entry (prim, X, Y). Likewise, for each query $X \leftarrow \Pi^{-1}(Y)$ we store an entry (prim, X, Y).
- For each query $C \leftarrow \textsc{TprpEnc}(i, T, M)$ we store a corresponding entry $(\mathsf{enc}, i, T, M, C)$. Likewise, for each query $M \leftarrow \textsc{TprpDec}(i, T, C)$ we store an entry $(\mathsf{enc}, i, T, M, C)$.

Due to the assumption that the adversary doesn't make redundant queries, there are no duplicate entries. We say that a transcript is *bad* if one of the following conditions happen:

1. There are entries $(\mathsf{enc}, i, T, M, C)$ and (prim, X, Y) such that either $M = X \oplus H(K_i, T)$ or $C = Y \oplus H(K_i, T)$.
2. There are entries $(\mathsf{enc}, i, T, M, C)$ and $(\mathsf{enc}, j, T^*, M^*, C^*)$ such that $i \neq j$ and either $M \oplus H(K_i, T) = M^* \oplus H(K_j, T^*)$ or $C \oplus H(K_i, T) = C^* \oplus H(K_j, T^*)$.
3. There are entries $(\mathsf{enc}, i, T, M, C)$ and $(\mathsf{enc}, i, T^*, M^*, C^*)$ such that either $M \oplus H(K_i, T) = M^* \oplus H(K_i, T^*)$ or $C \oplus H(K_i, T) = C^* \oplus H(K_i, T^*)$.

Such transcripts are likely to reveal the inconsistency in the ideal world. A transcript is *valid* if it can happen in the ideal world with non-zero probability. If a valid transcript is not bad then it is *good*.

Probability of bad transcripts. Let $\mathcal{T}_0$ be the random variable for the transcript in the ideal world. We now bound the probability that $\mathcal{T}_0$ is good. We will fix the queries and answers of the adversary, but still treat the revealed keys as random. That is, we are dealing with the conditional probability that $\mathcal{T}_0$ is bad, given a fixed choice of the queries and answers of the adversary. Our bound holds for any such choice, and thus it also holds for the unconditional probability that $\mathcal{T}_0$ is bad. For each $i \in \{1, 2, 3\}$, let Bad_i be the set of transcripts that violate the i-th constraint of badness. Then from the union bound,

$$\Pr[\mathcal{T}_0 \text{ is bad}] = \Pr[\mathcal{T}_0 \in (\mathsf{Bad}_1 \cup \mathsf{Bad}_2 \cup \mathsf{Bad}_3)] \le \sum_{i=1}^{3} \Pr[\mathcal{T}_0 \in \mathsf{Bad}_i] \, .$$

We first bound $\Pr[\mathcal{T}_0 \in \mathsf{Bad}_1]$. Consider entries $(\mathsf{enc}, i, T, M, C)$ and (prim, X, Y). Since H is δ-uniform, the chance that $H(K_i, T) \in \{M \oplus X, C \oplus Y\}$ is at most 2δ. Summing over pq possible pairs,

$$\Pr[\mathcal{T}_0 \in \mathsf{Bad}_1] \le 2pq \cdot \delta \, .$$

Next, we bound $\Pr[\mathcal{T}_0 \in \mathsf{Bad}_2]$. Consider entries $(\mathsf{enc}, i, T, M, C)$ and $(\mathsf{enc}, j, T^*, M^*, C^*)$ such that $i \ne j$. Since H is δ-uniform, the chance that $H(K_i, T) \in \{H(K_j, T^*) \oplus C \oplus C^*, H(K_j, T^*) \oplus M \oplus M^*\}$ is at most 2δ. Summing over at most $q^2/2$ possible pairs,

$$\Pr[\mathcal{T}_0 \in \mathsf{Bad}_2] \le q^2 \cdot \delta \, .$$

Finally, we bound $\Pr[\mathcal{T}_0 \in \mathsf{Bad}_3]$. Consider two different entries $(\mathsf{enc}, i, T, M, C)$ and $(\mathsf{enc}, i, T^*, M^*, C^*)$. From the definition and the assumption that the adversary doesn't make redundant queries, we must have $(M, T) \ne (M^*, T^*)$ and $(C, T) \ne (C^*, T^*)$. If $T = T^*$ then we can't have $M \oplus H(K_i, T) = M^* \oplus H(K_i, T^*)$ or $C \oplus H(K_i, T) = C^* \oplus H(K_i, T^*)$, so assume that $T \ne T^*$. Since H is ϵ-AXU, the chance that $H(K_i, T) \oplus H(K_i, T^*) \in \{M \oplus M^*, C \oplus C^*\}$ is at most 2ϵ. If there are u users in which user i makes B_i PRP queries then the number of such pairs are at most

$$\sum_{i=1}^{u} \frac{(B_i)^2}{2} \le \sum_{i=1} \frac{B_i B}{2} = \frac{qB}{2} \,.$$

Summing over those $qB/2$ possible pairs,

$$\Pr[\mathcal{T}_0 \in \mathsf{Bad}_3] \le qB \cdot \epsilon \,.$$

Hence, totally,

$$\Pr[\mathcal{T}_0 \text{ is bad}] \le \sum_{i=1}^{3} \Pr[\mathcal{T}_0 \in \mathsf{Bad}_i] \le qB \cdot \epsilon + q(q+2p) \cdot \delta \,.$$

Transcript ratio. Fix a good transcript θ. Suppose that there are u users and r pairs of (user, tweak) in θ. Let S_i be the set of PRP queries for the i-th user-tweak pair. Note that $|S_1| + \cdots + |S_u| = q$. In the ideal world, the event $\mathcal{T}_0 = \theta$ can be factored into three independent events:

- The revealed key of each user i is the same as the key K_i in θ. This happens with probability $1/|\mathsf{Keys}|^u$.
- The answers of the ideal-permutation queries match those given in θ. This happens with probability
$$\frac{1}{2^n(2^n-1)\cdots(2^n-p+1)} \,.$$
- The answers of the PRP queries match those given in θ. This happens with probability
$$\prod_{i=1}^{r} \frac{1}{2^n(2^n-1)\cdots(2^n-|S_i|+1)} \le \frac{1}{2^n(2^n-1)\cdots(2^n-q+1)} \,.$$

Hence

$$\Pr[\mathcal{T}_0 = \theta] \le \frac{1}{|\mathsf{Keys}|^u} \cdot \frac{1}{2^n \cdots (2^n-p+1)} \cdot \frac{1}{2^n \cdots (2^n-q+1)} \,.$$

For the real world, we factor the event $\mathcal{T}_1 = \theta$ into three events:

1. The revealed key of each user i is the same as the key K_i in θ. This happens with probability $1/|\mathsf{Keys}|^u$.
2. The answers of the ideal-permutation queries match those given in θ. Conditioned on the prior event, this happens with conditional probability
$$\frac{1}{2^n(2^n-1)\cdots(2^n-p+1)}\,.$$
3. The answers of the PRP queries match those given in θ. This translates to q calls for Π. Since θ is good, there is no conflict with prior ideal-cipher queries. Thus conditioned on the two prior events, this happens with conditional probability at least
$$\frac{1}{(2^n-p)(2^n-p-1)\cdots(2^n-p-q+1)} \ge \frac{1}{2^n(2^n-1)\cdots(2^n-q+1)}\,.$$

Hence
$$\Pr[\mathcal{T}_1 = \theta] \ge \frac{1}{|\mathsf{Keys}|^u}\cdot\frac{1}{2^n\cdots(2^n-p+1)}\cdot\frac{1}{2^n\cdots(2^n-q+1)}\,.$$
In other words,
$$\frac{\mathsf{p}_{\mathsf{S}_1}(\theta)}{\mathsf{p}_{\mathsf{S}_0}(\theta)} \ge 1\,.$$

Wrapping up. From Lemma 25,
$$\mathsf{Adv}^{\mathrm{tprp}}_{\tilde{E}}(\mathcal{A}) \le qB\cdot\epsilon + q(q+2p)\cdot\delta\,.$$

□

3.12 AXU Hashes

Direct Product Universal Hashing. Theorem 26 shows that if G is ϵ-AXU, then its d-direct product hash G^d is ϵ^d-AXU. This direct product construction is folklore [63, 88], and Theorem 7.4 in [63] proves this bound for $d = 2$.

Proposition 26. *Let* $\mathsf{G} : \{0,1\}^h \times \mathcal{X} \to \{0,1\}^\eta$ *be a* ϵ*-AXU hash function. For* $d \geq 1$*, define the* d*-direct product hash* $\mathsf{G}^d : \{0,1\}^{dh} \times \mathcal{X} \to \{0,1\}^{d\eta}$ *with a* dh*-bit key and* $d\eta$*-bit output as*

$$\mathsf{G}^d((H_1, \ldots, H_d), X) := \mathsf{G}(H_1, X) \parallel \cdots \parallel \mathsf{G}(H_d, X)\,. \tag{3.5}$$

Then G^d *is* ϵ^d*-AXU. That is, for all distinct inputs* $X, Y \in \mathcal{X}$ *and all outputs* $(Z_1, \ldots, Z_d) \in \{0,1\}^{d\eta}$*, it holds that*

$$\Pr_{(H_1,\ldots,H_d) \leftarrow\$ \{0,1\}^{dh}}[\mathsf{G}^d_{(H_1,\ldots,H_d)}(X) \oplus \mathsf{G}^d_{(H_1,\ldots,H_d)}(Y) = (Z_1, \ldots, Z_d)] \leq \epsilon^d\,.$$

Proof. Since the d subkeys H_i are sampled uniformly at random from $\{0,1\}^{dh}$, we can equivalently sample each of the d subkeys H_i independently and uniformly at random from $\{0,1\}^h$. Similarly, we can rewrite the winning event as the intersection of d events, to get

$$\Pr_{\forall i\ H_i \leftarrow\$ \{0,1\}^h}\left[\bigwedge_{i=1}^{d}(\mathsf{G}_{H_i}(X) \oplus \mathsf{G}_{H_i}(Y) = Z_i)\right]\,.$$

Since each of the d subkeys H_i is picked independently and uniformly at random, these d events are also mutually independent, so their joint probability is the product

$$\prod_{i=1}^{d}\left(\Pr_{H_i \leftarrow\$ \{0,1\}^h}[\mathsf{G}_{H_i}(X) \oplus \mathsf{G}_{H_i}(Y) = Z_i]\right)\,.$$

The final result follows from applying the fact that G is ϵ-AXU. □

The POLYVAL hash function. POLYVAL [91] is defined over a finite field F with

2^{128} elements defined with the irreducible polynomial

$$f(x) := x^{128} + x^{127} + x^{126} + x^{121} + 1$$

as $F := \mathrm{GF}(2^{128})[X]/f(x)$. Elements of F are polynomials $a_0 + a_1x + a_2x^2 + \cdots + a_{127}x^{127}$ with binary coefficients $a_i \in \mathrm{GF}(2)$. These polynomials can be represented as 128-bit strings $a_0 \cdots a_{127}$, with a_0 being the least significant bit of the first byte and a_{127} being the most significant bit of the last byte.

Given two field elements $a, b \in F$, their sum $a \oplus b$ is the bitwise xor of their coefficients, and their product $a \otimes b$ is polynomial multiplication modulo the irreducible polynomial. In addition, define the *dot* operation

$$\begin{aligned} a \bullet b &:= a \otimes b \otimes x^{-128} \\ &= a \otimes b \otimes (x^{127} + x^{124} + x^{121} + x^{114} + 1)\,. \end{aligned}$$

Given a sequence of field elements $X = (X_1, \ldots, X_s)$, its block-length is the number of field elements s, and it can be represented by a bitstring of length $128s$ by concatenating the 128-bit representations of all its field elements.

Definition 27 ([91]). $\mathsf{POLYVAL} : F \times F^* \to F$*, is defined as*

$$\mathsf{POLYVAL}(H, X_1, \ldots, X_s) := \sum_{i=1}^{s} X_i \bullet H_{(s+1-i)}\,,$$

where $H_{(j)} := H^j \otimes x^{-128\cdot(j-1)}$ *for* $j = 1, \ldots, s$.

Lemma 28 (Lemma 3 in [91]). *Let* $s_{\max}$ *be an upper bound on the block-length of messages* $\mathsf{POLYVAL}$ *is queried on. Equivalently, let* $\ell_{\max} = 128s_{\max}$ *be the maximum length (in bits) that* $\mathsf{POLYVAL}$ *is queried on. Then,* $\mathsf{POLYVAL}$ *is* ϵ*-AXU with*

$$\epsilon = \frac{s_{\max}}{2^{128}} = \frac{\ell_{\max}}{2^{135}}\,.$$

The direct product POLYVAL hash function. We combine Theorem 26 and Theorem 28 to construct and evaluate the security of d-direct product POLYVAL.

Definition 29. $\mathsf{POLYVAL}^d : F^d \times F^* \to F^d$*, is defined as*

$$\begin{aligned}&\mathsf{POLYVAL}^d((H_1, \dots, H_d), X_1, \dots, X_s) \\ &\quad := \mathsf{POLYVAL}(H_1, X_1, \dots, X_s) \parallel \cdots \parallel \mathsf{POLYVAL}(H_d, X_1, \dots, X_s)\,.\end{aligned}$$

Corollary 30. *Let* $s_{\max}$ *be an upper bound on the block-length of messages* $\mathsf{POLYVAL}^d$ *is queried on. Equivalently, let* $\ell_{\max} = 128 s_{\max}$ *be the maximum length (in bits) that* $\mathsf{POLYVAL}$ *is queried on. Then,* $\mathsf{POLYVAL}^d$ *is* ϵ*-AXU with*

$$\epsilon = \frac{(s_{\max})^d}{2^{128d}} = \frac{(\ell_{\max})^d}{2^{135d}}\,.$$

3.13 More Performance Graphs

We provide cycles-per-byte graphs for Intel Raptor Lake (Figure 3.26) and ARM Cortex-A76 (Figure 3.27). Apple does not provide access to cycle counters, so we omit a graph for the M2 Pro.

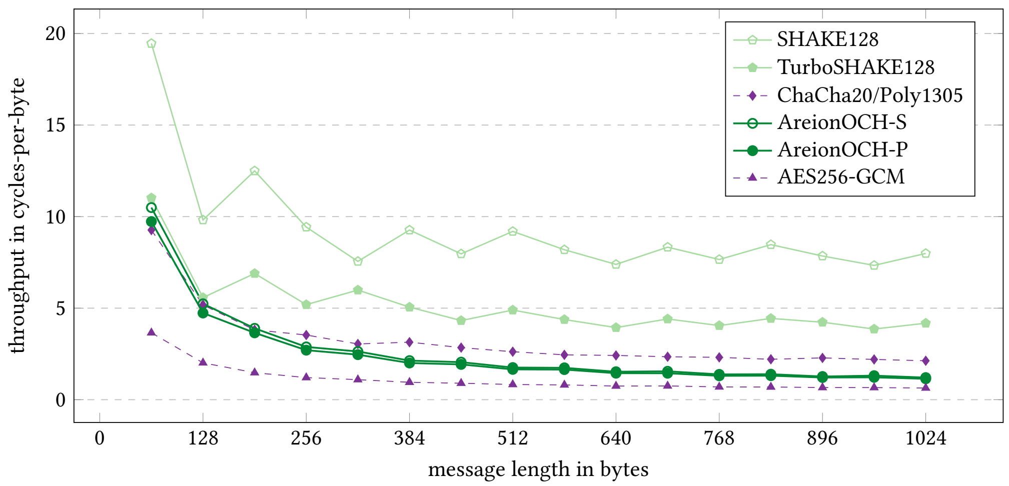

Figure 3.26: **AEAD performance on desktop x86-64.** Throughput in CPU cycles-per-byte (y-axis, lower is faster) for encrypting messages of various sizes (x-axis, in bytes) with 13 bytes of associated data on an Intel Raptor Lake processor. Solid lines indicate schemes that achieve 128-bit NAE and 128-bit CMT security, and dashed lines indicate schemes that do not.

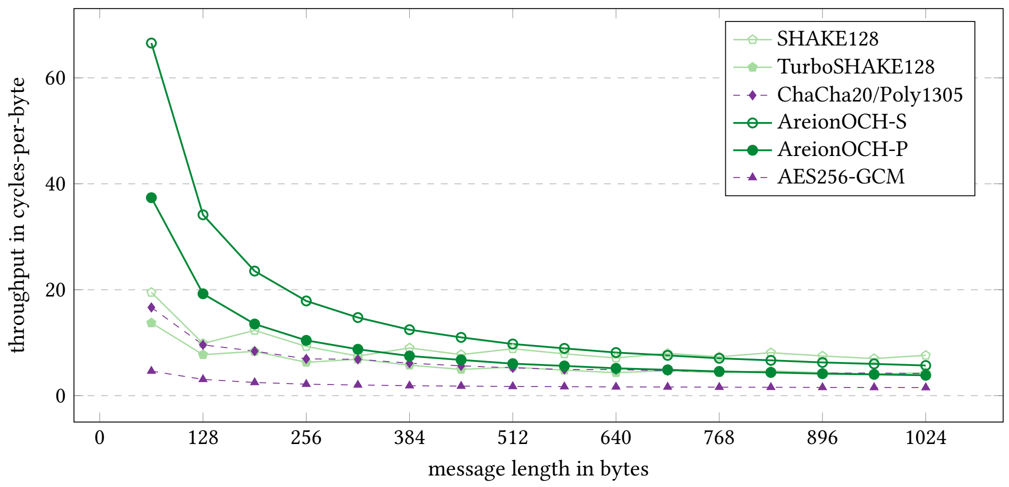

Figure 3.27: **AEAD performance on lightweight arm64.** Throughput in CPU cycles-per-byte (y-axis, lower is faster) for encrypting messages of various sizes (x-axis, in bytes) with 13 bytes of associated data on an ARM Cortex-A76 processor. Solid lines indicate schemes that achieve 128-bit NAE and 128-bit CMT security, and dashed lines indicate schemes that do not.

CHAPTER 4

FLEXIBLE AUTHENTICATED ENCRYPTION[*]

In this work we propose a new evolution of symmetric encryption, what we refer to as flexible AEAD. We think this will facilitate and enable upcoming standardization efforts. We give definitions for flexible AEAD as well as a concrete realization in the form of a scheme we simply call Flex. First we set the stage.

Emerging goals for AEAD. Recall that in a scheme for AEAD (Authenticated Encryption with Associated Data) [144], encryption takes key, nonce, associated data and message to deterministically return a ciphertext. The classical security requirement is unique-nonce AE (UNAE) security. This means privacy of the message, and authenticity of the message and associated data, assuming encryption never reuses a nonce. The NIST standard AES-GCM [73] is an example of an AEAD scheme.

Since the standardization of AES-GCM, developers and researchers have identified a number of further, desirable security and operational attributes for AEAD. Security attributes include committing security [87, 71, 78, 3, 15, 52], misuse resistance [149] and optional nonce-hiding [22]; operational attributes include parallelizability, robustness in the sense of [95] and support for arbitrary-length nonces and keys. These and other attributes are part of a comprehensive list in the IETF draft on properties of AEAD algorithms [45]. Let us now expand on (some of) these attributes and why they are desirable.

Committing security. A recent line of work has demonstrated the need for key commitment for AEAD schemes. Key commitment asks that it be hard to find a ciphertext

[*]This chapter is joint work with Julia Len, Viet Tung Hoang, Mihir Bellare, and Thomas Ristenpart. Earlier versions of this work were presented at Third NIST Workshop on Block Cipher Modes of Operation [127] and at Real World Crypto 2024 [32].

that decrypts correctly under two (or more) different adversarially-chosen keys [78, 87]. Non-key-committing AEAD was first shown to be a problem for moderation in encrypted messaging [71, 87], and later in password-based encryption [119] and envelope encryption [3], among others. These findings have pushed the cryptography community to begin proposing [3] and deploying new key-committing AEAD schemes [3, 11]. Indeed the recent IETF draft on properties of AEAD algorithms explicitly cites key commitment as a security goal [45, §4.3.3].

Key commitment definitions don't model attacks in which adversaries exploit lack of commitment to the associated data or nonce. Bellare and Hoang [15] introduced the notion of *context commitment*, and recent work by Menda et al. [125] highlights that many AEAD schemes, including some that achieve key commitment, do not achieve context commitment. This motivates the need to expand our goal to context commitment.

Unfortunately, no currently standardized schemes achieve context commitment. This suggests we need to define and standardize new AEAD schemes.

<u>Misuse resistance.</u> Stipulating non-repeating nonces is easy in theory but harder to ensure in practice, where errors and misconfigurations have been reported to lead to repeating nonces. For many widely used and standardized UNAE AEAD schemes, this has led to damaging attacks [42]. Misuse-resistant AE (MRAE) [149] mitigates this by providing UNAE-security when nonces do not repeat, plus as good as possible security if they do. Standardization of an MRAE scheme is a desirable goal.

<u>Optional nonce-hiding.</u> Embedded in the syntax and usage of AEAD is a weakness relating to the way nonces are handled. Namely, since the nonce is needed for decryption, it is sent in the clear unless the receiver already has it. But, as Bellare, Ng, and

Tackmann [22] point out, some choices of nonces compromise message privacy (for both UNAE and MRAE) and others (like counters) compromise sender anonymity. To remedy this, they proposed modifying the security definition of AE to have mandatory nonce-hiding (termed AE2 [22]). AE2 is indeed necessary for security in some contexts like anonymous AE [50]. However, mandating nonce-hiding is wasteful in the widespread context of random nonces, and and Bellare and Hoang [17] proposed a further refinement AE3 that mandates optional nonce hiding (they split the nonce into a public nonce and a secret nonce, and mandate nonce-hiding for the secret nonce.) This optional nonce-hiding emerges as another desirable attribute for a standard.

Robustness. The ciphertext expansion of a scheme is defined as the difference between ciphertext length and plaintext length. UNAE security requires some ciphertext expansion. (Usually 128 bits for AES-GCM.) But, applications cannot permit this, for example, communication-constrained applications like 5G want short expansion to achieve lower latency, and in-place fixed-size applications like disk encryption and database encryption want no expansion so the encrypted content can fit in the same sector or field. The answer is robust AE [95], where one gets the best security possible with a given constraint on the ciphertext expansion.

Key and nonce lengths. The 96-bit nonce-length of AES-GCM is viewed as a limitation because with random nonces it permits at most 2^{48} encryptions before a key change is needed. Schemes would ideally have either no maximum nonce length or a very large one. Similarly, different users or applications want different security levels and thus different key sizes. The need for post-quantum security is also pushing key sizes for symmetric cryptography up. Ideally, a scheme should handle keys of different and arbitrary lengths.

The challenge for standardization. One approach for standardization would be to standardize a different scheme for each of some choice of goals. However, the dimensions indicated above give rise to rather a lot of goals. Indeed, there are two choices for AE security (UNAE or MRAE), then a choice for committing security (yes or no) and another for AE2 (yes or no), already $2^3 = 8$ goals, with further possible choices for robustness, parallelizability, streaming support, and key and nonce lengths.

An alternative is to standardize a scheme for the strongest goal (for example, MRAE, committing and nonce-hiding) but this will mean that applications needing less stringent attributes will suffer a unnecessary yet costly penalty. Indeed, one can't unfortunately achieve all the goals simultaneously while providing best-possible speed for each individual goal. Misuse resistance requires a full pass over the plaintext before the first bits of ciphertext can be produced, and known robust AE construction approaches require building a length-preserving enciphering scheme and then combining with the encode-then-encipher paradigm [23]. In either case you cannot have a scheme that is streaming, let alone parallelizable.

With flexibility, we suggest a different route.

Flexible AEAD. At a high level, our idea is to reformulate AEAD so that it takes as run-time input a configuration. The configuration, denoted `cfg` throughout, specifies an operating mode. But the same secret key can be safely used across different configurations, even on a message-by-message basis. So one can encrypt one message under a secret key using a fully parallelizable configuration, and another message under an MRAE configuration, with the same secret key. Protocol developers can use this flexibility as needed for their applications. In most cases they will presumably use a single configuration; in this case flexible AEAD provides some defense-in-depth with respect to misconfigurations for the same secret key.

We provide new definitions to capture the syntax, semantics, and security of flexible AEAD. Flexible AEAD will start with some mandatory (always provided) security and operational attributes. Then through choices of the above-discussed configuration, it will further provide other attributes as options. Yet, this will be implemented in a unified way with what is essentially a single scheme.

As mandatory, we ask for classic UNAE and context commitment. The reason for making the latter mandatory is to avoid errors arising from developers not knowing that they need to turn it on. (Indeed, the errors and attacks we have seen are arising from applications assuming implicitly that commitment is present.) We also always ask for the ability to handle arbitrary-length keys and nonces. As optional, if requested in cfg, we support providing MRAE, nonce-hiding, and streaming capability. Other configurations can be added as needed.

The Flex scheme. Towards realizing flexible AEAD, we propose a clean-slate approach that reimagines AEAD scheme construction to allow it to provide modern security and performance for a broad range of application domains. Our Flex scheme conceptually has a straightforward structure: use a key derivation function (KDF) to derive sub-keys that can be safely used with different underlying schemes. Thus constructively, Flex, brings into the AEAD what used to be external, namely, a KDF function.

This flexibility would perhaps seem to require a complex AEAD scheme, depending on many underlying mechanisms. But we build a “wide waist” strategy: one scheme has a variety of different configurations all using the same underlying primitives. In particular, we build all of Flex’s various configurations from a single underlying cryptographic permutation. For instance, we use a realization of tweaked Even-Mansour [56] built from a permutation. While Flex works for any sufficiently wide permutation, we have mostly focused on instantiations using Simpira [92].

We propose three primary Flex configurations. The OCH configuration provides an OCB-like mode of operation that is fully parallelizable, but not MR nor robust. It is the most performant. We also provide CIV (committing synthetic IV) an SIV [150, 149] variant that provides MR security, and is as parallelizable as possible. Finally, we aim to provide robust AE based on prior constructions [93, 95] using our underlying tweakable block cipher.

Organization. The rest of the chapter is organized as follows. In Section 4.1 we define our new cryptographic primitive flexible AEAD (F-AEAD). Then, in Section 4.2 we provide an overview of our F-AEAD construction called Flex and the three initial Flex configurations.

4.1 Flexible AEAD Definitions

We start with new definitions for AEAD. These build off a variety of prior work, as we explain further below, but weave in new aspects that we think will benefit secure, flexible deployability and usage.

Basic notation. We let $B = \{0, 1\}^8$ be the set of 8-bit bytes. To be better aligned with practice, we will define primitives over byte strings rather than bit strings, but note that our constructions can easily be changed to handle arbitrary bit strings.

Elements of F-AEAD. Recall that in classical AEAD [151, 144], encryption $C \leftarrow \mathsf{Enc}(K, N, A, M)$ takes key, nonce, AD and message to return a ciphertext, while decryption $M \leftarrow \mathsf{Dec}(K, N, A, C)$ takes key, nonce, AD and ciphertext to return the message. Security can be basic (unique-nonce) [151, 144], requiring that encryption under a key not repeat a nonce, or advanced (misuse resistant) [149], requiring only that it not repeat nonce, AD and message.

We provide a slightly different and more general formalization called flexible AEAD (F-AEAD). A novel element of our framework is the introduction of an extra initialization algorithm Init which chooses the appropriate AEAD algorithm based on the parameters specified in a configuration input cfg. For instance, amongst other things, cfg includes a flag $\mathsf{cfg.mr}$ that says whether or not misuse resistance is requested. This allows a user to dynamically make this choice in different settings without changing the key.

F-AEAD syntax. Formally, a flexible AEAD (F-AEAD) scheme FAE specifies several algorithms and associated quantities, as follows:

- $K_* \leftarrow\$ \mathsf{FAE.Kg}(\kappa)$: The randomized key generation algorithm takes an integer $\kappa \in \mathsf{FAE.KL}$, which is the key length (in bytes), and outputs a byte string $K_* \in B^\kappa$ called a (secret) key. Here $\mathsf{FAE.KL} \subseteq \mathbb{N}$ is the set of allowed key lengths.
- $L \leftarrow \mathsf{FAE.Init}(\mathsf{cfg}, K_*, AD_s)$: The deterministic initialization algorithm initializes a new configuration based on parameters specified in the algorithm. It takes as input a configuration cfg, key K_*, and setup associated data $AD_s \in B^*$. The scheme specifies a set $\mathsf{FAE.Cfgs}$ of possible (allowed) configurations, and it is required that cfg belongs to this set. The algorithm outputs a (secret) subkey $L \in \mathsf{FAE.Cfgs} \times B^*$, where the first component is the configuration and the second component is the encryption key, or it outputs the distinguished error symbol $\perp$. If $L \neq \perp$, its encryption key component has length $\mathsf{FAE.SL}(\mathsf{cfg}, |K_*|)$, where $\mathsf{FAE.SL}$ is called the subkey-length function. The configuration associated with L is specified by $L.\mathsf{cfg}$.
- $C \leftarrow \mathsf{FAE.Enc}(L, N, AD_m, M)$: The deterministic encryption algorithm takes as

input subkey L, nonce $N \in B^*$, message associated data $AD_m \in B^*$, and message $M \in B^*$, and returns a ciphertext $C \in B^* \cup \{\bot\}$. If $C \neq \bot$, it has length $\mathsf{FAE.CL}(K.\mathsf{cfg}, |N|, |M|)$, where $\mathsf{FAE.CL}$ is called the ciphertext-length function.

- $M \leftarrow \mathsf{FAE.Dec}(L, \tilde{N}, AD_m, C)$: The decryption algorithm takes as input subkey L, nonce $\tilde{N}$, associated data AD_m, and ciphertext C, and returns a message $M \in B^* \cup \{\bot\}$ that is either a byte string or the distinguished error symbol $\bot$. If nonce-hiding is desirable, then $\tilde{N}$ may be set to the empty string ε.

We proceed to some remarks, explanations, and choices.

Configurations. Configurations are customizable, but a configuration cfg always includes flags $\mathsf{cfg.mr}, \mathsf{cfg.nh} \in \{\mathsf{true}, \mathsf{false}\}$ indicating whether or not misuse resistant security or nonce-hiding is requested and $\mathsf{cfg.perm} \in B^*$ indicating the type of permutation to use. These flags may be set independently of each other.

Flag	Configuration	Type
mr	misuse resistance	boolean
nh	nonce-hiding	boolean
perm	permutation	string

Many key lengths. Typically, an encryption scheme mandates a single length for its key. Our definition instead allows keys of different lengths. (In our construction, any length up to some maximum.) Security will depend on the shortest key length used. The usual setting of a single key length is captured by having $\mathsf{AEAD.KL}$ be a singleton set. (It contains just one length.)

Validity of inputs. Whether or not $\mathsf{FAE.Enc}(L, N, AD_m, M)$ returns $\bot$ is required to depend only on $L.\mathsf{cfg}, |L|, |AD_m|, |M|$ and can be easily computed given these quantities. We require that $\mathsf{FAE.Enc}(L, N, AD_m, M) = \perp$ if $L.\mathsf{cfg} \notin \mathsf{FAE.Cfgs}$. Furthermore, if $L.\mathsf{cfg.mr} = \mathsf{true}$, meaning that misuse resistance is requested, then we require that $\mathsf{FAE.Dec}(L, \tilde{N}, AD_m, C) = \perp$ if $\tilde{N} \neq \varepsilon$. This is because specifying a nonce during decryption when nonce misuse is requested should indicate an error.

Correctness. For correctness, we require that decryption of legitimate ciphertexts always succeeds. In detail, for any L, N, AD_m, M, if $C \leftarrow \mathsf{FAE.Enc}(L, N, AD_m, M) \neq \perp$ and $L.\mathsf{cfg.mr} = \mathsf{false}$, then $\mathsf{FAE.Dec}(L, N, AD_m, C) = M$. Otherwise, if $L.\mathsf{cfg.mr} = \mathsf{true}$, then it should be that $\mathsf{FAE.Dec}(L, \varepsilon, AD_m, C) = M$.

Tidiness. Extending [149], we say that FAE is *tidy* if: For any L, N, AD_m, C, if $M \leftarrow \mathsf{FAE.Dec}(L, N, AD_m, C) \neq \perp$ then $\mathsf{FAE.Enc}(L, N, AD_m, M) = C$. This is not always required, but our schemes provide it.

Ciphertext length. Typically the ciphertext is stretched by some constant τ as compared with the plaintext message. But we may have schemes for which the configuration dictates different security levels, and in turn config-dependent stretch.

4.2 The Flex AEAD

We propose an F-AEAD called Flex. Flex offers solutions to common deployment settings and security across different configurations, while retaining performance that is competitive with or beats the best known (non-flexible) AEAD schemes. To do so, Flex's design is modular, and can be viewed as baking into the AEAD's design what is traditionally left to users of AEAD, namely choice of different AEAD types (misuse-resistance versus not) and key derivation. We will show how this affords Flex to easily adapt to application requirements, while maintaining programmatic configuration for developers with associated simple-to-use APIs.

Overview. We build Flex modularly. At a high level there are four main components all built from the same underlying permutation π:

- A key derivation function (KDF) that achieves CR-PRF security. It is used to derive a subkey from the secret subkey, associated data, and configuration. By default this is a simple sponge-style hash construction, but we also give modes that allow parallelization and precomputation of static elements of the input, such as the associated data .
- A non-misuse resistant encryption mode called OCH that is an OCB-like mode that provides context committing security and nonce-hiding while being fully parallelizable.
- A misuse resistant encryption mode called CIV that is an SIV-like mode that provides context committing security and nonce-hiding while being maximally parallelizable.
- A robust encryption mode.

All components are new to this work, but of course use some design elements seen in prior work as we will highlight where relevant. We can then mix-and-match the different KDF modes with the different encryption modes to provide various designs, which enables a wide variety of AEAD schemes. At this level our individual constructions follow the Hash-then-Encrypt mechanism from Bellare and Hoang [15] that transforms a key-committing AEAD into a context-committing AEAD. But we carefully arrange our constructions, down to the level of using a fast-to-rekey specially constructed Even-Mansour [76] cipher, to avoid inefficiencies.

Preliminaries. Let $\pi : \{0,1\}^n \to \{0,1\}^n$ be a permutation that we will model as an ideal permutation. Fix a number $\tau < n$. We will target security of $\tau/2$ bits or greater. As a running example we will use $\tau = 256$ and $n = 512$, giving 128-bit security.

A permutation-based function. We use a few basic primitives built from π. The first is a variable-output-length function $F : \{0,1\}^* \times \mathbb{Z} \times \{0,1\}^* \to \{0,1\}^*$. We define that $F(K, \ell, X)$ outputs a bit string of length ℓ using key K. We will require that F be a collision-resistant (CR) pseudorandom function (PRF). One possibility for F is to use a sponge-type construction [36] with the desired output length prepended to the message. In this work, we provide our own construction that minimizes the underlying calls to π. For simplicity, we do not describe the details of this construction here.

A permutation-based TBC. Our second underlying primitive is a tweakable block-cipher (TBC), also built from π. Let $\mathcal{N} = \{0,1\}^{\leq n-8}$ be the set of all bit strings whose length is at most $n - 8$. Then we use a tweakable blockcipher $\tilde{E} : \mathsf{Keys} \times \mathsf{Tweaks} \times \{0,1\}^n \to \{0,1\}^n$ whose key space is $\mathsf{Keys} = \{0,1\}^\tau \times \{0,1\}^\tau \times \{0,1\}^\tau$, tweak space is $\mathsf{Tweaks} = \{0, 1, (1,1), (1,2)\} \cup \left(\mathbb{Z} \times [1,2,3] \times \mathcal{N}\right)$ and whose block length is n bits. The set of possible nonces includes any bit string N with $0 \leq |N| \leq n - 1$.

At a high level, $\tilde{E}$ is constructed using a special version of tweaked Even-Mansour [56] built from a permutation. For simplicity, we will not cover the details of the construction here.

4.2.1 Flex Overview

We provide a top-down description of Flex. It supports up to 256 different configurations, allowing cfg to be represented by one byte. For clarity in our presentation, in addition to the configuration flags we specify in Section 4.1, we let $\mathsf{cfg.TL}$ indicate the number of bits of tag we desire, $\mathsf{cfg.NL}$ indicate the length of the nonce, and $\mathsf{cfg.SL}$ indicate value τ such that the subkey is length 3τ. We define Flex as follows.

- $\mathsf{Flex.Kg}(\kappa)$ outputs a uniform bit string K_* of length κ.
- $\mathsf{Flex.Init}(\mathsf{cfg}, K_*, AD_s)$ verifies that cfg belongs to the set of possible (allowed) configurations $\mathsf{FAE.Cfgs}$ and that $|K_*|$ is in the set of allowed key lengths $\mathsf{FAE.KL}$. If not, it outputs an error. Otherwise, it applies the key derivation function $\mathsf{KD}(K_*, \mathsf{cfg}, AD_s)$ to generate three keys (L_1, L_2, L_3) each of length $\tau = \mathsf{cfg.SL}$ bits and forms subkey $L \leftarrow (\mathsf{cfg}, L_1 \parallel L_2 \parallel L_3)$. We thus define $\mathsf{FAE.SL}(\mathsf{cfg}, |K_*|)$ as returning $3 \cdot \mathsf{cfg.SL}$. Optionally it may perform some precomputation of values that will help speed-up encryption (e.g., the table L used for fast computation of our TBC). We refer to such values that get used by multiple encryption calls as state, and notate it by st.
- $\mathsf{Flex.Enc}(L, N, AD_m, M)$ works in different ways depending on $L.\mathsf{cfg}$. For instance, if $L.\mathsf{cfg.mr} = \mathsf{true}$, then misuse resistant mode CIV is chosen for encryption; otherwise, non-misuse resistant mode OCH is chosen. The output is a ciphertext consisting of a triple (C_{hdr}, C, T) consisting of a (possibly empty) ciphertext header C_{hdr}, a ciphertext body C, and a tag T and an updated state st. Some modes may have C_{hdr} empty.
- $\mathsf{Flex.Dec}(L, \tilde{N}, AD_m, C)$, like encryption, proceeds depending on the parameters specified in $L.\mathsf{cfg}$. If $L.\mathsf{cfg.nh} = \mathsf{true}$ and $\tilde{N} \neq \varepsilon$, then decryption returns an error. Otherwise, it proceeds according to the mode specified by the configuration to return the message M.

4.2.2 The Flex KDF

We start by describing the KDF used by Flex, which simply combines an encoding of the inputs $(\mathsf{cfg}, K_*, AD_s)$ and generates subkeys L_1, L_2, L_3 and (optionally) some pre-computed state.

We compute an encoding K' of the key K_* to serve as the PRF key. In particular, if Flex.KL is a singleton set (only one allowed key length), then we let K' be K_*. Otherwise, we let K' be a 64-bit encoding of the key length followed by the key: $\langle |K_*| \rangle_{64} \parallel K_*$. The bit string encoding S is computed as follows. The first byte of S is cfg. Similar to how K_* is encoded, if cfg allows only a single setup AD length, then we append AD_s and otherwise append a 64-bit encoding of the length and then the AD. All of this ensures an unambiguous encoding of cfg and AD_s. We then apply $F(K', 3\tau, S)$ to yield an encryption subkey triple (L_1, L_2, L_3). In our scheme, L_1 will serve as a commitment to the input.

4.2.3 Flex Modes

We briefly describe the three modes provided by Flex.

The non-misuse resistant configuration OCH. Flex chooses OCH used when the configuration specifies that misuse-resistance is not required via $\mathsf{cfg.mr} = \mathsf{false}$. It is context committing and has optional nonce-hiding. It is formally defined and analyzed in Chapter 3.

The MR configuration CIV. CIV is the misuse resistant mode associated with Flex. It is used when the configuration specifies that MR is requested via $\mathsf{cfg.mr} = \mathsf{true}$. We briefly give an overview of how it works here. More details on its construction will be forthcoming.

At a high level, the construction first computes a PRF based on PMAC [40] over the message using the subkey. It computes a CR hash over the output of the PRF and the subkey to generate the authentication tag, which serves as the synthetic IV. Finally, it uses a CTR-mode with the subkey to encrypt the message and form the ciphertext.

The robust configuration. As we mention in the introduction, another desirable functionality for Flex is to provide a robust AEAD mode. Robust AEAD schemes allow specifying the maximal ciphertext stretch, where ciphertext integrity should be achieved up to the maximal possible.

We aim to support robust AEAD by using a prior TBC, such as encode-then-encipher with an EME* [93] mode or AEZ [95]. Like our MR configuration, more details on this construction will be forthcoming.

BIBLIOGRAPHY

[1] Michel Abdalla, Mihir Bellare, and Gregory Neven. “Robust Encryption”. In: *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*. Ed. by Daniele Micciancio. Vol. 5978. Lecture Notes in Computer Science. Springer, 2010, pp. 480–497. DOI: 10.1007/978-3-642-11799-2_28.

[2] Ange Albertini et al. “How to Abuse and Fix Authenticated Encryption Without Key Commitment”. In: *USENIX Security 2022*. Ed. by Kevin R. B. Butler and Kurt Thomas. USENIX Association, Aug. 2022, pp. 3291–3308. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/albertini.

[3] Ange Albertini et al. *How to Abuse and Fix Authenticated Encryption Without Key Commitment*. USENIX Security 2022. 2022. URL: https://ia.cr/2020/1456.

[4] Moreno Ambrosin et al. *Tink*. 2021. URL: https://github.com/google/tink/releases/tag/v1.6.1.

[5] Moreno Ambrosin et al. *Tink EAX Key Manager*. 2021. URL: https://github.com/google/tink/blob/v1.6.1/java_src/src/main/java/com/google/crypto/tink/aead/AesEaxKeyManager.java#L115-L116.

[6] Elena Andreeva et al. “APE: Authenticated Permutation-Based Encryption for Lightweight Cryptography”. In: *FSE 2014*. Ed. by Carlos Cid and Christian Rechberger. Vol. 8540. LNCS. Springer, Berlin, Heidelberg, Mar. 2015, pp. 168–186. DOI: 10.1007/978-3-662-46706-0_9.

[7] Elena Andreeva et al. “Security of Keyed Sponge Constructions Using a Modular Proof Approach”. In: *FSE 2015*. Ed. by Gregor Leander. Vol. 9054. LNCS. Springer, Berlin, Heidelberg, Mar. 2015, pp. 364–384. DOI: 10.1007/978-3-662-48116-5_18.

[8] Scott Arciszewski. *Galois Extended Mode.* NIST Workshop on the Requirements for an Accordion Cipher Mode. 2024. URL: https://csrc.nist.gov/csrc/media/Events/2024/accordion-cipher-mode-workshop-2024/documents/papers/galois-extended-mode.pdf.

[9] Scott Arciszewski. *XChaCha: eXtended-nonce ChaCha and AEAD_XChaCha20_Poly1305.* Internet-Draft draft-irtf-cfrg-xchacha-03. Work in Progress. Internet Engineering Task Force, Jan. 2020. 18 pp. URL: https://datatracker.ietf.org/doc/draft-irtf-cfrg-xchacha/03/.

[10] Jean-Philippe Aumasson et al. "BLAKE2: Simpler, Smaller, Fast as MD5". In: *ACNS 2013.* Ed. by Michael J. Jacobson Jr. et al. Vol. 7954. LNCS. Springer, Berlin, Heidelberg, June 2013, pp. 119–135. DOI: 10.1007/978-3-642-38980-1_8.

[11] AWS. *Supported algorithm suites in the AWS Encryption SDK*. Accessed 2021-09-23. 2021. URL: https://docs.aws.amazon.com/encryption-sdk/latest/developer-guide/supported-algorithms.html.

[12] Zhenzhen Bao et al. "Automatic Search of Meet-in-the-Middle Preimage Attacks on AES-like Hashing". In: *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I.* Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12696. Lecture Notes in Computer Science. Springer, 2021, pp. 771–804. DOI: 10.1007/978-3-030-77870-5_27.

[13] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. "Pseudorandom Functions Revisited: The Cascade Construction and Its Concrete Security". In: *37th FOCS.* IEEE Computer Society Press, Oct. 1996, pp. 514–523. DOI: 10.1109/SFCS.1996.548510.

[14] Mihir Bellare, Hannah Davis, and Felix Günther. "Separate Your Domains: NIST PQC KEMs, Oracle Cloning and Read-Only Indifferentiability". In: *EUROCRYPT 2020, Part II*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12106. LNCS. Springer, Cham, May 2020, pp. 3–32. DOI: 10.1007/978-3-030-45724-2_1.

[15] Mihir Bellare and Viet Tung Hoang. "Efficient Schemes for Committing Authenticated Encryption". In: *EUROCRYPT 2022, Part II*. Ed. by Orr Dunkelman and Stefan Dziembowski. Vol. 13276. LNCS. Springer, Cham, May 2022, pp. 845–875. DOI: 10.1007/978-3-031-07085-3_29.

[16] Mihir Bellare and Viet Tung Hoang. "Efficient Schemes for Committing Authenticated Encryption". In: *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part II*. Ed. by Orr Dunkelman and Stefan Dziembowski. Vol. 13276. Lecture Notes in Computer Science. Springer, 2022, pp. 845–875. DOI: 10.1007/978-3-031-07085-3_29.

[17] Mihir Bellare and Viet Tung Hoang. "Succinctly-Committing Authenticated Encryption". In: *CRYPTO 2024, Part IV*. Ed. by Leonid Reyzin and Douglas Stebila. Vol. 14923. LNCS. Springer, Cham, Aug. 2024, pp. 305–339. DOI: 10.1007/978-3-031-68385-5_10.

[18] Mihir Bellare, Viet Tung Hoang, and Cong Wu. *The Landscape of Committing Authenticated Encryption*. The Third NIST Workshop on Block Cipher Modes of Operation. 2023. URL: https://csrc.nist.gov/Presentations/2023/landscape-of-committing-authenticated-encryption.

[19] Mihir Bellare and Chanathip Namprempre. "Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm". In: *J. Cryptol.* 21.4 (2008), pp. 469–491. DOI: 10.1007/s00145-008-9026-x.

[20] Mihir Bellare and Chanathip Namprempre. “Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm”. In: *Journal of Cryptology* 21.4 (Oct. 2008), pp. 469–491. DOI: 10.1007/s00145-008-9026-x.

[21] Mihir Bellare and Chanathip Namprempre. “Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm”. In: *ASIACRYPT 2000*. Ed. by Tatsuaki Okamoto. Vol. 1976. LNCS. Springer, Berlin, Heidelberg, Dec. 2000, pp. 531–545. DOI: 10.1007/3-540-44448-3_41.

[22] Mihir Bellare, Ruth Ng, and Björn Tackmann. “Nonces Are Noticed: AEAD Revisited”. In: *CRYPTO 2019, Part I*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11692. LNCS. Springer, Cham, Aug. 2019, pp. 235–265. DOI: 10.1007/978-3-030-26948-7_9.

[23] Mihir Bellare and Phillip Rogaway. “Encode-Then-Encipher Encryption: How to Exploit Nonces or Redundancy in Plaintexts for Efficient Cryptography”. In: *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*. Ed. by Tatsuaki Okamoto. Vol. 1976. Lecture Notes in Computer Science. Springer, 2000, pp. 317–330. DOI: 10.1007/3-540-44448-3_24. URL: https://cseweb.ucsd.edu/~mihir/papers/ee.pdf.

[24] Mihir Bellare and Phillip Rogaway. “Encode-Then-Encipher Encryption: How to Exploit Nonces or Redundancy in Plaintexts for Efficient Cryptography”. In: *ASIACRYPT 2000*. Ed. by Tatsuaki Okamoto. Vol. 1976. LNCS. Springer, Berlin, Heidelberg, Dec. 2000, pp. 317–330. DOI: 10.1007/3-540-44448-3_24.

[25] Mihir Bellare and Phillip Rogaway. *Introduction to Modern Cryptography*. 2005. URL: https://web.cs.ucdavis.edu/~rogaway/classes/227/spring05/book/main.pdf.

[26] Mihir Bellare and Phillip Rogaway. “The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs”. In: *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*. Ed. by Serge Vaudenay. Vol. 4004. Lecture Notes in Computer Science. Springer, 2006, pp. 409–426. DOI: 10.1007/11761679_25.

[27] Mihir Bellare and Phillip Rogaway. “The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs”. In: *EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. LNCS. Springer, Berlin, Heidelberg, May 2006, pp. 409–426. DOI: 10.1007/11761679_25.

[28] Mihir Bellare, Phillip Rogaway, and David A. Wagner. “The EAX Mode of Operation”. In: *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*. Ed. by Bimal K. Roy and Willi Meier. Vol. 3017. Lecture Notes in Computer Science. Springer, 2004, pp. 389–407. DOI: 10.1007/978-3-540-25937-4_25. URL: https://web.cs.ucdavis.edu/~rogaway/papers/eax.pdf.

[29] Mihir Bellare and Laura Shea. “Flexible Password-Based Encryption: Securing Cloud Storage and Provably Resisting Partitioning-Oracle Attacks”. In: *CT-RSA 2023*. Ed. by Mike Rosulek. Vol. 13871. LNCS. Springer, Cham, Apr. 2023, pp. 594–621. DOI: 10.1007/978-3-031-30872-7_23.

[30] Mihir Bellare and Björn Tackmann. “The Multi-user Security of Authenticated Encryption: AES-GCM in TLS 1.3”. In: *CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. LNCS. Springer, Berlin, Heidelberg, Aug. 2016, pp. 247–276. DOI: 10.1007/978-3-662-53018-4_10.

[31] Mihir Bellare et al. “Ask Your Cryptographer if Context-Committing AEAD Is Right for You”. In: *Presented at Real World Crypto Symposium*. 2023.

[32] Mihir Bellare et al. *Building the Next Generation of AEAD*. Presented at Real World Crypto 2024. 2024.

[33] Mihir Bellare et al. "Format-Preserving Encryption". In: *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers*. Ed. by Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini. Vol. 5867. Lecture Notes in Computer Science. Springer, 2009, pp. 295–312. DOI: 10.1007/978-3-642-05445-7_19.

[34] Daniel J Bernstein et al. "ChaCha, a variant of Salsa20". In: *Workshop record of SASC*. Vol. 8. 2008, pp. 3–5.

[35] Daniel J. Bernstein. "The Poly1305-AES Message-Authentication Code". In: *FSE 2005*. Ed. by Henri Gilbert and Helena Handschuh. Vol. 3557. LNCS. Springer, Berlin, Heidelberg, Feb. 2005, pp. 32–49. DOI: 10.1007/11502760_3.

[36] Guido Bertoni et al. "Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications". In: *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*. Ed. by Ali Miri and Serge Vaudenay. Vol. 7118. Lecture Notes in Computer Science. Springer, 2011, pp. 320–337. DOI: 10.1007/978-3-642-28496-0_19.

[37] Guido Bertoni et al. "Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications". In: *SAC 2011*. Ed. by Ali Miri and Serge Vaudenay. Vol. 7118. LNCS. Springer, Berlin, Heidelberg, Aug. 2012, pp. 320–337. DOI: 10.1007/978-3-642-28496-0_19.

[38] Guido Bertoni et al. “Farfalle: parallel permutation-based cryptography”. In: *IACR Trans. Symm. Cryptol.* 2017.4 (2017), pp. 1–38. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i4.1-38.

[39] Guido Bertoni et al. “Sponge functions”. In: *ECRYPT hash workshop*. Vol. 2007. 2007.

[40] John Black and Phillip Rogaway. “A Block-Cipher Mode of Operation for Parallelizable Message Authentication”. In: *EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Vol. 2332. LNCS. Springer, Berlin, Heidelberg, Apr. 2002, pp. 384–397. DOI: 10.1007/3-540-46035-7_25.

[41] John Black, Phillip Rogaway, and Thomas Shrimpton. “Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV”. In: *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*. Ed. by Moti Yung. Vol. 2442. Lecture Notes in Computer Science. Springer, 2002, pp. 320–335. DOI: 10.1007/3-540-45708-9_21.

[42] Hanno Böck et al. *Nonce-Disrespecting Adversaries: Practical Forgery Attacks on GCM in TLS*. WOOT 16. 2016. URL: https://eprint.iacr.org/2016/475.

[43] BoringSSL Authors. *speed.cc*. 2025. URL: https://github.com/google/boringssl/blob/e056f59c7dfdcf891af03bc7900c946ac485c78f/tool/speed.cc.

[44] Priyanka Bose, Viet Tung Hoang, and Stefano Tessaro. “Revisiting AES-GCM-SIV: Multi-user Security, Faster Key Derivation, and Better Bounds”. In: *EUROCRYPT 2018, Part I*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10820. LNCS. Springer, Cham, Apr. 2018, pp. 468–499. DOI: 10.1007/978-3-319-78381-9_18.

[45] Andrey Bozhko. *Properties of AEAD algorithms*. Internet-Draft draft-irtf-cfrg-aead-properties-01. Work in Progress. Internet Engineering Task Force, Mar. 2023. 15 pp. URL: https://datatracker.ietf.org/doc/draft-irtf-cfrg-aead-properties/01/.

[46] CAESAR committee. *CAESAR call for submissions, draft 1*. 2013. URL: http://competitions.cr.yp.to/caesar-call-1.html.

[47] Larry Carter and Mark N. Wegman. "Universal Classes of Hash Functions (Extended Abstract)". In: *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*. Ed. by John E. Hopcroft, Emily P. Friedman, and Michael A. Harrison. ACM, 1977, pp. 106–112. DOI: 10.1145/800105.803400. URL: https://doi.org/10.1145/800105.803400.

[48] Avik Chakraborti et al. "Light-OCB: parallel lightweight authenticated cipher with full security". In: *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer. 2021, pp. 22–41.

[49] Debrup Chakraborty and Palash Sarkar. "A General Construction of Tweakable Block Ciphers and Different Modes of Operations". In: *Information Security and Cryptology*. Ed. by Helger Lipmaa, Moti Yung, and Dongdai Lin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 88–102. ISBN: 978-3-540-49610-6. DOI: 10.1007/11937807_8.

[50] John Chan and Phillip Rogaway. "Anonymous AE". In: *ASIACRYPT 2019, Part II*. Ed. by Steven D. Galbraith and Shiho Moriai. Vol. 11922. LNCS. Springer, Cham, Dec. 2019, pp. 183–208. DOI: 10.1007/978-3-030-34621-8_7.

[51] John Chan and Phillip Rogaway. "On Committing Authenticated-Encryption". In: *European Symposium on Research in Computer Security*. Springer. 2022, pp. 275–294. URL: https://ia.cr/2022/1260.

[52] John Chan and Phillip Rogaway. "On Committing Authenticated-Encryption". In: *ESORICS 2022, Part II*. Ed. by Vijayalakshmi Atluri et al. Vol. 13555. LNCS. Springer, Cham, Sept. 2022, pp. 275–294. DOI: 10.1007/978-3-031-17146-8_14.

[53] Donghoon Chang et al. "A Keyed Sponge Construction with Pseudorandomness in the Standard Model". en. In: N/A, Washington, DC, US, 2012-03-22 00:03:00 2012. URL: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=910823.

[54] Shan Chen and John P. Steinberger. "Tight Security Bounds for Key-Alternating Ciphers". In: *EUROCRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. LNCS. Springer, Berlin, Heidelberg, May 2014, pp. 327–350. DOI: 10.1007/978-3-642-55220-5_19.

[55] Yu Long Chen et al. *Key Committing Security of AEZ*. The Third NIST Workshop on Block Cipher Modes of Operation. 2023. URL: https://csrc.nist.gov/Presentations/2023/key-committing-security-of-aez.

[56] Benoit Cogliati, Rodolphe Lampe, and Yannick Seurin. "Tweaking Even-Mansour Ciphers". In: *CRYPTO 2015, Part I*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9215. LNCS. Springer, Berlin, Heidelberg, Aug. 2015, pp. 189–208. DOI: 10.1007/978-3-662-47989-6_9.

[57] Valve Corporation. *Steam Hardware & Software Survey: March 2025*. steampowered.com. 2025. URL: https://web.archive.org/web/20250402020128/https://store.steampowered.com/hwsurvey/Steam-Hardware-Software-Survey-Welcome-to-Steam.

[58] Joan Daemen, Silvia Mella, and Gilles Van Assche. *Committing authenticated encryption based on SHAKE*. Cryptology ePrint Archive, Report 2023/1494. 2023. URL: https://eprint.iacr.org/2023/1494.

[59] Joan Daemen, Bart Mennink, and Gilles Van Assche. “Full-State Keyed Duplex with Built-In Multi-user Support”. In: *ASIACRYPT 2017, Part II*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10625. LNCS. Springer, Cham, Dec. 2017, pp. 606–637. DOI: 10.1007/978-3-319-70697-9_21.

[60] Joan Daemen et al. *Shaking up authenticated encryption*. Cryptology ePrint Archive, Report 2024/1618. 2024. URL: https://eprint.iacr.org/2024/1618.

[61] Joan Daemen et al. “The design of Xoodoo and Xoofff”. In: *IACR Trans. Symm. Cryptol.* 2018.4 (2018), pp. 1–38. ISSN: 2519-173X. DOI: 10.13154/tosc.v2018.i4.1-38.

[62] Joan Daemen et al. “Xoodyak, a lightweight cryptographic scheme”. In: *IACR Trans. Symm. Cryptol.* 2020.S1 (2020), pp. 60–87. ISSN: 2519-173X. DOI: 10.13154/tosc.v2020.iS1.60-87.

[63] Jean Paul Degabriele et al. “The Security of ChaCha20-Poly1305 in the Multi-User Setting”. In: *ACM CCS 2021*. Ed. by Giovanni Vigna and Elaine Shi. ACM Press, Nov. 2021, pp. 1981–2003. DOI: 10.1145/3460120.3484814.

[64] Frank Denis et al. *AEAD constructions, Robustness*. 2022. URL: https://doc.libsodium.org/secret-key%5C_cryptography/aead#robustness.

[65] Frank Denis and Samuel Lucas. *The AEGIS Family of Authenticated Encryption Algorithms*. Internet-Draft draft-irtf-cfrg-aegis-aead-16. Work in Progress. Internet Engineering Task Force, Feb. 2025. 73 pp. URL: https://datatracker.ietf.org/doc/draft-irtf-cfrg-aegis-aead/16/.

[66] Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. “Cryptanalysis of Simpira v1”. In: *SAC 2016*. Ed. by Roberto Avanzi and Howard M. Heys. Vol. 10532. LNCS. Springer, Cham, Aug. 2016, pp. 284–298. DOI: 10.1007/978-3-319-69453-5_16.

[67] Christoph Dobraunig et al. *Ascon v1.2*. Submission to NIST. 2021. URL: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/ascon-spec-final.pdf.

[68] Christoph Dobraunig et al. "Ascon v1.2: Lightweight Authenticated Encryption and Hashing". In: *Journal of Cryptology* 34.3 (July 2021), p. 33. DOI: 10.1007/s00145-021-09398-9.

[69] Christoph Dobraunig et al. *Ascon v1.2: Submission to NIST*. May 2021. URL: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/ascon-spec-final.pdf.

[70] Christoph Dobraunig et al. *Schwaemm and Esch: Lightweight Authenticated Encryption and Hashing using the Sparkle Permutation Family v1.1*. Submission to NIST. 2019. URL: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/sparkle-spec-round2.pdf.

[71] Yevgeniy Dodis et al. "Fast Message Franking: From Invisible Salamanders to Encryptment". In: *CRYPTO 2018, Part I*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10991. LNCS. Springer, Cham, Aug. 2018, pp. 155–186. DOI: 10.1007/978-3-319-96884-1_6.

[72] Thai "thaidn" Duong. *AWS: In-band protocol negotiation and robustness weaknesses in AWS KMS and Encryption SDKs*. CVE-2020-8897. 2020. URL: https://github.com/google/security-research/security/advisories/GHSA-wqgp-vphw-hphf.

[73] Morris Dworkin. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. Tech. rep. NIST Special Publication (SP)

NIST SP 800-38D. Gaithersburg, MD: National Institute of Standards and Technology, 2007. DOI: 10.6028/NIST.SP.800-38D.

[74] Morris Dworkin. *Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality.* NIST Special Publication 800-38C. 2004. DOI: 10.6028/NIST.SP.800-38C.

[75] Morris Dworkin. *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication.* NIST Special Publication 800-38B. 2005. DOI: 10.6028/NIST.SP.800-38B.

[76] Shimon Even and Yishay Mansour. "A Construction of a Cipher From a Single Pseudorandom Permutation". In: *ASIACRYPT'91.* Ed. by Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto. Vol. 739. LNCS. Springer, Berlin, Heidelberg, Nov. 1993, pp. 210–224. DOI: 10.1007/3-540-57332-1_17.

[77] Facebook. *Messenger Secret Conversations: Technical Whitepaper.* 2017. URL: https://about.fb.com/wp-content/uploads/2016/07/messenger-secret-conversations-technical-whitepaper.pdf.

[78] Pooya Farshim, Claudio Orlandi, and Razvan Rosie. "Security of Symmetric Primitives under Incorrect Usage of Keys". In: *IACR Trans. Symmetric Cryptol.* 2017.1 (2017), pp. 449–473. DOI: 10.13154/tosc.v2017.i1.449-473.

[79] Pooya Farshim, Claudio Orlandi, and Răzvan Roşie. "Security of Symmetric Primitives under Incorrect Usage of Keys". In: *IACR Trans. Symm. Cryptol.* 2017.1 (2017), pp. 449–473. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i1.449-473.

[80] Pooya Farshim et al. "Robust Encryption, Revisited". In: *Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26 - March 1, 2013. Proceedings.* Ed. by

Kaoru Kurosawa and Goichiro Hanaoka. Vol. 7778. Lecture Notes in Computer Science. Springer, 2013, pp. 352–368. DOI: 10.1007/978-3-642-36362-7_22.

[81] Peter Gazi, Krzysztof Pietrzak, and Stefano Tessaro. "The Exact PRF Security of Truncation: Tight Bounds for Keyed Sponges and Truncated CBC". In: *CRYPTO 2015, Part I*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9215. LNCS. Springer, Berlin, Heidelberg, Aug. 2015, pp. 368–387. DOI: 10.1007/978-3-662-47989-6_18.

[82] Virgil D. Gligor and Pompiliu Donescu. "Fast Encryption and Authentication: XCBC Encryption and XECB Authentication Modes". In: *FSE 2001*. Ed. by Mitsuru Matsui. Vol. 2355. LNCS. Springer, Berlin, Heidelberg, Apr. 2002, pp. 92–108. DOI: 10.1007/3-540-45473-X_8.

[83] Michel Goemans. *Chernoff bounds, and some applications*. 2015. URL: https://math.mit.edu/~goemans/18310S15/chernoff-notes.pdf.

[84] Shafi Goldwasser and Mihir Bellare. *Lecture Notes on Cryptography*. 2008. URL: https://cseweb.ucsd.edu/~mihir/papers/gb.pdf.

[85] Robert Granger et al. "Improved Masking for Tweakable Blockciphers with Applications to Authenticated Encryption". In: *EUROCRYPT 2016, Part I*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9665. LNCS. Springer, Berlin, Heidelberg, May 2016, pp. 263–293. DOI: 10.1007/978-3-662-49890-3_11.

[86] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. "Message Franking via Committing Authenticated Encryption". In: *CRYPTO 2017, Part III*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10403. LNCS. Springer, Cham, Aug. 2017, pp. 66–97. DOI: 10.1007/978-3-319-63697-9_3.

[87] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. "Message Franking via Committing Authenticated Encryption". In: *Advances in Cryptology - CRYPTO 2017*

- 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10403. Lecture Notes in Computer Science. Springer, 2017, pp. 66–97. DOI: 10.1007/978-3-319-63697-9_3.

[88] Shay Gueron. Personal communication. 2024.

[89] Shay Gueron. *Double Nonce Derive Key AES-GCM (DNDK-GCM)*. Internet-Draft draft-gueron-cfrg-dndkgcm-02. Work in Progress. Internet Engineering Task Force, Mar. 2025. 40 pp. URL: https://datatracker.ietf.org/doc/draft-gueron-cfrg-dndkgcm/02/.

[90] Shay Gueron. *Key Committing AEADs*. Cryptology ePrint Archive, Report 2020/1153. 2020. URL: https://eprint.iacr.org/2020/1153.

[91] Shay Gueron, Adam Langley, and Yehuda Lindell. *AES-GCM-SIV: Specification and Analysis*. Cryptology ePrint Archive, Report 2017/168. 2017. URL: https://eprint.iacr.org/2017/168.

[92] Shay Gueron and Nicky Mouha. "Simpira v2: A Family of Efficient Permutations Using the AES Round Function". In: *ASIACRYPT 2016, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. LNCS. Springer, Berlin, Heidelberg, Dec. 2016, pp. 95–125. DOI: 10.1007/978-3-662-53887-6_4.

[93] Shai Halevi. "EME*: Extending EME to Handle Arbitrary-Length Messages with Associated Data". In: *INDOCRYPT 2004*. Ed. by Anne Canteaut and Kapalee Viswanathan. Vol. 3348. LNCS. Springer, Berlin, Heidelberg, Dec. 2004, pp. 315–327. DOI: 10.1007/978-3-540-30556-9_25.

[94] Dan Harkins. *Synthetic Initialization Vector (SIV) Authenticated Encryption Using the Advanced Encryption Standard (AES)*. Request for Comments - Informational. RFC. 2008. URL: https://datatracker.ietf.org/doc/rfc5297/.

[95] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. "Robust Authenticated-Encryption AEZ and the Problem That It Solves". In: *EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Berlin, Heidelberg, Apr. 2015, pp. 15–44. DOI: 10.1007/978-3-662-46800-5_2.

[96] Viet Tung Hoang and Stefano Tessaro. "Key-Alternating Ciphers and Key-Length Extension: Exact Bounds and Multi-user Security". In: *CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. LNCS. Springer, Berlin, Heidelberg, Aug. 2016, pp. 3–32. DOI: 10.1007/978-3-662-53018-4_1.

[97] Viet Tung Hoang, Stefano Tessaro, and Aishwarya Thiruvengadam. "The Multi-user Security of GCM, Revisited: Tight Bounds for Nonce Randomization". In: *ACM CCS 2018*. Ed. by David Lie et al. ACM Press, Oct. 2018, pp. 1429–1440. DOI: 10.1145/3243734.3243816.

[98] Seth Hoffert. *Nonce- and Redundancy-encrypting Modes with Farfalle*. Cryptology ePrint Archive, Paper 2022/1711. 2022. URL: https://eprint.iacr.org/2022/1711.

[99] Akiko Inoue and Kazuhiko Minematsu. "Parallelizable Authenticated Encryption with Small State Size". In: *SAC 2019*. Ed. by Kenneth G. Paterson and Douglas Stebila. Vol. 11959. LNCS. Springer, Cham, Aug. 2019, pp. 618–644. DOI: 10.1007/978-3-030-38471-5_25.

[100] Takanori Isobe and Mostafizar Rahman. *Key Committing Security Analysis of AEGIS*. Cryptology ePrint Archive, Report 2023/1495. 2023. URL: https://eprint.iacr.org/2023/1495.

[101] Takanori Isobe and Mostafizar Rahman. *Key Committing Security Analysis of AEGIS*. Cryptology ePrint Archive, Paper 2023/1495. 2023. URL: https://eprint.iacr.org/2023/1495.

[102] Takanori Isobe et al. “Areion: Highly-Efficient Permutations and Its Applications to Hash Functions for Short Input”. In: *IACR TCHES* 2023.2 (2023), pp. 115–154. DOI: 10.46586/tches.v2023.i2.115-154.

[103] Jérémy Jean. “Cryptanalysis of Haraka”. In: *IACR Trans. Symm. Cryptol.* 2016.1 (2016), pp. 1–12. ISSN: 2519-173X. DOI: 10.13154/tosc.v2016.i1.1-12. URL: https://tosc.iacr.org/index.php/ToSC/article/view/531.

[104] Jérémy Jean et al. “The Deoxys AEAD Family”. In: *Journal of Cryptology* 34.3 (July 2021), p. 31. DOI: 10.1007/s00145-021-09397-w.

[105] Philipp Jovanovic et al. “Beyond Conventional Security in Sponge-Based Authenticated Encryption Modes”. In: *Journal of Cryptology* 32.3 (July 2019), pp. 895–940. DOI: 10.1007/s00145-018-9299-7.

[106] Charanjit S. Jutla. “Encryption Modes with Almost Free Message Integrity”. In: *EUROCRYPT 2001.* Ed. by Birgit Pfitzmann. Vol. 2045. LNCS. Springer, Berlin, Heidelberg, May 2001, pp. 529–544. DOI: 10.1007/3-540-44987-6_32.

[107] Panos Kampanakis et al. *Practical Challenges with AES-GCM and the need for a new cipher.* Third NIST Workshop on Block Cipher Modes of Operation. 2023. URL: https://csrc.nist.gov/csrc/media/Events/2023/third-workshop-on-block-cipher-modes-of-operation/documents/accepted-papers/Practical%20Challenges%20with%20AES-GCM.pdf.

[108] Joe Kilian and Phillip Rogaway. “How to Protect DES Against Exhaustive Key Search (an Analysis of DESX)”. In: *J. Cryptol.* 14.1 (2001), pp. 17–35. DOI: 10.1007/s001450010015.

[109] Stefan Kölbl et al. “Haraka v2 - Efficient Short-Input Hashing for Post-Quantum Applications”. In: *IACR Trans. Symmetric Cryptol.* 2016.2 (2016), pp. 1–29. DOI: 10.13154/tosc.v2016.i2.1-29.

[110] Hugo Krawczyk. “New Hash Functions For Message Authentication”. In: *EUROCRYPT’95*. Ed. by Louis C. Guillou and Jean-Jacques Quisquater. Vol. 921. LNCS. Springer, Berlin, Heidelberg, May 1995, pp. 301–310. DOI: 10.1007/3-540-49264-X_24.

[111] Hugo Krawczyk. *The OPAQUE Asymmetric PAKE Protocol.* Tech. rep. Oct. 2019. URL: https://datatracker.ietf.org/doc/draft-krawczyk-cfrg-opaque/03/.

[112] Ted Krovetz and Phillip Rogaway. “The Design and Evolution of OCB”. In: *Journal of Cryptology* 34.4 (Oct. 2021), p. 36. DOI: 10.1007/s00145-021-09399-8.

[113] Ted Krovetz and Phillip Rogaway. *The OCB Authenticated-Encryption Algorithm.* Request for Comments - Informational. RFC. 2014. URL: https://datatracker.ietf.org/doc/rfc7253/.

[114] Ted Krovetz and Phillip Rogaway. “The Software Performance of Authenticated-Encryption Modes”. In: *FSE 2011*. Ed. by Antoine Joux. Vol. 6733. LNCS. Springer, Berlin, Heidelberg, Feb. 2011, pp. 306–327. DOI: 10.1007/978-3-642-21702-9_18.

[115] Kaoru Kurosawa. “Power of a Public Random Permutation and Its Application to Authenticated Encryption”. In: *IEEE Transactions on Information Theory* 56.10 (2010), pp. 5366–5374.

[116] Kaoru Kurosawa. *Power of a Public Random Permutation and its Application to Authenticated-Encryption.* Cryptology ePrint Archive, Report 2002/127. 2002. URL: https://eprint.iacr.org/2002/127.

[117] Julia Len, Paul Grubbs, and Thomas Ristenpart. “Authenticated Encryption with Key Identification”. In: *ASIACRYPT 2022, Part III*. Ed. by Shweta Agrawal and

Dongdai Lin. Vol. 13793. LNCS. Springer, Cham, Dec. 2022, pp. 181–209. DOI: 10.1007/978-3-031-22969-5_7.

[118] Julia Len, Paul Grubbs, and Thomas Ristenpart. “Partitioning Oracle Attacks”. In: *USENIX Security 2021*. Ed. by Michael Bailey and Rachel Greenstadt. USENIX Association, Aug. 2021, pp. 195–212. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/len.

[119] Julia Len, Paul Grubbs, and Thomas Ristenpart. “Partitioning Oracle Attacks”. In: *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*. Ed. by Michael Bailey and Rachel Greenstadt. USENIX Association, 2021, pp. 195–212. URL: https://ia.cr/2020/1491.

[120] Moses Liskov, Ronald L. Rivest, and David Wagner. “Tweakable Block Ciphers”. In: *CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. LNCS. Springer, Berlin, Heidelberg, Aug. 2002, pp. 31–46. DOI: 10.1007/3-540-45708-9_3.

[121] Raspberry Pi Ltd. *Raspberry Pi 5 Datasheet*. raspberrypi.com. 2025. URL: https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf.

[122] Atul Luykx, Bart Mennink, and Kenneth G. Paterson. “Analyzing Multi-key Security Degradation”. In: *ASIACRYPT 2017, Part II*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10625. LNCS. Springer, Cham, Dec. 2017, pp. 575–605. DOI: 10.1007/978-3-319-70697-9_20.

[123] Ueli M. Maurer. “Indistinguishability of Random Systems”. In: *EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Vol. 2332. LNCS. Springer, Berlin, Heidelberg, Apr. 2002, pp. 110–132. DOI: 10.1007/3-540-46035-7_8.

[124] David A. McGrew and John Viega. “The Security and Performance of the Galois/Counter Mode (GCM) of Operation”. In: *INDOCRYPT 2004*. Ed. by Anne

Canteaut and Kapalee Viswanathan. Vol. 3348. LNCS. Springer, Berlin, Heidelberg, Dec. 2004, pp. 343–355. DOI: 10.1007/978-3-540-30556-9_27.

[125] Sanketh Menda et al. "Context Discovery and Commitment Attacks - How to Break CCM, EAX, SIV, and More". In: *EUROCRYPT 2023, Part IV*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14007. LNCS. Springer, Cham, Apr. 2023, pp. 379–407. DOI: 10.1007/978-3-031-30634-1_13.

[126] Sanketh Menda et al. *Context Discovery and Commitment Attacks: How to Break CCM, EAX, SIV, and More*. Cryptology ePrint Archive, Report 2023/526. 2023. URL: https://eprint.iacr.org/2023/526.

[127] Sanketh Menda et al. *Flexible Authenticated Encryption*. Third NIST Workshop on Block Cipher Modes of Operation. 2023. URL: https://csrc.nist.gov/csrc/media/Events/2023/third-workshop-on-block-cipher-modes-of-operation/documents/accepted-papers/Flexible%20Authenticated%20Encryption.pdf.

[128] Sanketh Menda et al. *The OCH Authenticated Encryption Scheme*. To appear at ACM CCS 2025. 2025.

[129] Bart Mennink. *Understanding the Duplex and Its Security*. Cryptology ePrint Archive, Report 2022/1340. 2022. URL: https://eprint.iacr.org/2022/1340.

[130] Bart Mennink. "XPX: Generalized Tweakable Even-Mansour with Improved Security Guarantees". In: *CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. LNCS. Springer, Berlin, Heidelberg, Aug. 2016, pp. 64–94. DOI: 10.1007/978-3-662-53018-4_3.

[131] Bart Mennink, Reza Reyhanitabar, and Damian Vizár. "Security of Full-State Keyed Sponge and Duplex: Applications to Authenticated Encryption". In: *ASIACRYPT 2015, Part II*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9453. LNCS.

Springer, Berlin, Heidelberg, Nov. 2015, pp. 465–489. DOI: 10.1007/978-3-662-48800-3_19.

[132] Nicky Mouha and Atul Luykx. “Multi-key Security: The Even-Mansour Construction Revisited”. In: *CRYPTO 2015, Part I*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9215. LNCS. Springer, Berlin, Heidelberg, Aug. 2015, pp. 209–223. DOI: 10.1007/978-3-662-47989-6_10.

[133] Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. “Reconsidering Generic Composition”. In: *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. Lecture Notes in Computer Science. Springer, 2014, pp. 257–274. DOI: 10.1007/978-3-642-55220-5_15.

[134] Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. “Reconsidering Generic Composition”. In: *EUROCRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. LNCS. Springer, Berlin, Heidelberg, May 2014, pp. 257–274. DOI: 10.1007/978-3-642-55220-5_15.

[135] Chanathip Namprempre, Phillip Rogaway, and Tom Shrimpton. *AE5 Security Notions: Definitions Implicit in the CAESAR Call*. Cryptology ePrint Archive, Report 2013/242. 2013. URL: https://eprint.iacr.org/2013/242.

[136] Yoav Nir and Adam Langley. *ChaCha20 and Poly1305 for IETF Protocols*. RFC 7539. May 2015. DOI: 10.17487/RFC7539. URL: https://www.rfc-editor.org/info/rfc7539.

[137] Yoav Nir and Adam Langley. *ChaCha20 and Poly1305 for IETF Protocols*. RFC 8439. June 2018. DOI: 10.17487/RFC8439. URL: https://www.rfc-editor.org/info/rfc8439.

[138] Jacques Patarin. “The “Coefficients H” Technique (Invited Talk)”. In: *SAC 2008*. Ed. by Roberto Maria Avanzi, Liam Keliher, and Francesco Sica. Vol. 5381. LNCS. Springer, Berlin, Heidelberg, Aug. 2009, pp. 328–345. DOI: 10.1007/978-3-642-04159-4_21.

[139] Thomas Pornin. *CPU Cycle Counter*. GitHub. 2025. URL: https://github.com/pornin/cycle-counter.

[140] John PreußMattsson, Ben Smeets, and Erik Thormarker. *Proposals for Standardization of Encryption Schemes*. The Third NIST Workshop on Block Cipher Modes of Operation. 2023. URL: https://csrc.nist.gov/csrc/media/Events/2023/third-workshop-on-block-cipher-modes-of-operation/documents/accepted-papers/Proposals%20for%20Standardization%20of%20Encryption%20Schemes%20Final.pdf.

[141] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Aug. 2018. DOI: 10.17487/RFC8446. URL: https://www.rfc-editor.org/info/rfc8446.

[142] Eric Rescorla, Hannes Tschofenig, and Nagendra Modadugu. *The Datagram Transport Layer Security (DTLS) Protocol Version 1.3*. RFC 9147. Apr. 2022. DOI: 10.17487/RFC9147. URL: https://www.rfc-editor.org/info/rfc9147.

[143] Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. “Careful with Composition: Limitations of Indifferentiability and Universal Composability”. In: *IACR Cryptol. ePrint Arch.* (2011), p. 339. URL: http://ia.cr/2011/339.

[144] Phillip Rogaway. “Authenticated-Encryption With Associated-Data”. In: *ACM CCS 2002*. Ed. by Vijayalakshmi Atluri. ACM Press, Nov. 2002, pp. 98–107. DOI: 10.1145/586110.586125.

[145] Phillip Rogaway. "Authenticated-encryption with associated-data". In: *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*. Ed. by Vijayalakshmi Atluri. ACM, 2002, pp. 98–107. DOI: 10.1145/586110.586125.

[146] Phillip Rogaway. "Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC". In: *ASIACRYPT 2004*. Ed. by Pil Joong Lee. Vol. 3329. LNCS. Springer, Berlin, Heidelberg, Dec. 2004, pp. 16–31. DOI: 10.1007/978-3-540-30539-2_2.

[147] Phillip Rogaway, Mihir Bellare, and John Black. "OCB: A block-cipher mode of operation for efficient authenticated encryption". In: *ACM Trans. Inf. Syst. Secur.* 6.3 (2003), pp. 365–403.

[148] Phillip Rogaway and Thomas Shrimpton. "A Provable-Security Treatment of the Key-Wrap Problem". In: Lecture Notes in Computer Science 4004 (2006). Ed. by Serge Vaudenay, pp. 373–390. DOI: 10.1007/11761679_23.

[149] Phillip Rogaway and Thomas Shrimpton. "A Provable-Security Treatment of the Key-Wrap Problem". In: *EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. LNCS. Springer, Berlin, Heidelberg, May 2006, pp. 373–390. DOI: 10.1007/11761679_23.

[150] Phillip Rogaway and Thomas Shrimpton. *The SIV Mode of Operation for Deterministic Authenticated-Encryption (Key Wrap) and Misuse-Resistant Nonce-Based Authenticated-Encryption*. Draft 0.32. 2007. URL: https://web.cs.ucdavis.edu/~rogaway/papers/siv.pdf.

[151] Phillip Rogaway et al. "OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption". In: *ACM CCS 2001*. Ed. by Michael K. Reiter and

Pierangela Samarati. ACM Press, Nov. 2001, pp. 196–205. DOI: 10.1145/501983.502011.

[152] Sondre Rønjom. "Invariant subspaces in Simpira". In: *IACR Cryptol. ePrint Arch.* (2016), p. 248. URL: http://eprint.iacr.org/2016/248.

[153] Markku-Juhani O. Saarinen and Jean-Philippe Aumasson. *The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)*. RFC 7693. Nov. 2015. DOI: 10.17487/RFC7693. URL: https://www.rfc-editor.org/info/rfc7693.

[154] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 10.6)*. 2025. URL: https://www.sagemath.org.

[155] Yu Sasaki et al. *Minalpher v1*. Submission to CAESAR competition. 2014. URL: https://info.isl.ntt.co.jp/crypt/minalpher/files/minalpherv1.pdf.

[156] Yu Sasaki et al. *Minalpher v1.1*. https://info.isl.ntt.co.jp/crypt/minalpher/files/minalpherv11.pdf. 2015.

[157] Sophie Schmieg. *Invisible Salamanders in AES-GCM-SIV*. 2020. URL: https://keymaterial.net/2020/09/07/invisible-salamanders-in-aes-gcm-siv/.

[158] Sophie, indistinguishable from random noise (@SchmiegSophie). via Twitter. 2020. URL: https://web.archive.org/web/20200909134511/https://twitter.com/SchmiegSophie/status/1303690812933382148.

[159] NIST Lightweight Cryptography Team. *NIST announces the selection of the Ascon family for lightweight cryptography standardization*. Feb. 2023. URL: https://www.nist.gov/news-events/news/2023/02/lightweight-cryptography-standardization-process-nist-selects-ascon.

[160] Martin Thomson and Sean Turner. *Using TLS to Secure QUIC*. RFC 9001. May 2021. DOI: 10.17487/RFC9001. URL: https://www.rfc-editor.org/info/rfc9001.

[161] Meltem Sönmez Turan et al. *Ascon-Based Lightweight Cryptography Standards for Constrained Devices*. Tech. rep. NIST Special Publication (SP) NIST SP 800-232 ipd. Gaithersburg, MD: National Institute of Standards and Technology, 2024. DOI: 10.6028/NIST.SP.800-232.ipd.

[162] Meltem Sönmez Turan et al. *Report on Lightweight Cryptography*. Tech. rep. NIST Internal Report 8114. Gaithersburg, MD: National Institute of Standards and Technology, 2017. DOI: 10.6028/NIST.IR.8114.

[163] Filippo Valsorda. *The XAES-256-GCM extended-nonce AEAD*. Community Cryptography Specification Project. 2024. URL: https://c2sp.org/XAES-256-GCM.

[164] David A. Wagner. "A Generalized Birthday Problem". In: *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*. Ed. by Moti Yung. Vol. 2442. Lecture Notes in Computer Science. Springer, 2002, pp. 288–303. URL: https://people.eecs.berkeley.edu/~daw/papers/genbday.html.

[165] Mark N. Wegman and Larry Carter. "New Hash Functions and Their Use in Authentication and Set Equality". In: *Journal of Computer and System Sciences* 22 (1981), pp. 265–279.

[166] Michael J. Wiener. "The Full Cost of Cryptanalytic Attacks". In: *J. Cryptol.* 17.2 (2004), pp. 105–124. DOI: 10.1007/s00145-003-0213-5.

[167] Hongjun Wu and Bart Preneel. "AEGIS: A Fast Authenticated Encryption Algorithm". In: *SAC 2013*. Ed. by Tanja Lange, Kristin Lauter, and Petr Lisonek. Vol. 8282. LNCS. Springer, Berlin, Heidelberg, Aug. 2014, pp. 185–201. DOI: 10.1007/978-3-662-43414-7_10.

[168] Ping Zhang and Honggang Hu. *On the Provable Security of the Tweakable Even-Mansour Cipher Against Multi-Key and Related-Key Attacks*. Cryptology ePrint Archive, Report 2016/1172. 2016. URL: https://eprint.iacr.org/2016/1172.