Breaking the IEEE Encryption Standard – XCB-AES in Two Queries

Amit Singh Bhati[1,2] and Elena Andreeva[3]

[1] COSIC, KU Leuven, Belgium,
[2] 3MI Labs, Belgium,
[3] TU Wien, Austria
amitsingh.bhati@3milabs.tech
elena.andreeva@tuwien.ac.at

Abstract. Tweakable enciphering modes (TEMs) provide security in various storage and space-critical applications, including disk and file-based encryption and packet-based communication protocols. XCB-AES (originally introduced as XCBv2) is specified in the IEEE 1619.2 standard for encryption of sector-oriented storage media and comes with a formal security proof for block-aligned messages.
In this work, we present the *first* plaintext recovery attack on XCB-AES – *the shared difference attack*, demonstrating that the security of XCB-AES is fundamentally flawed. Our plaintext recovery attack is highly efficient and requires only two queries (one enciphering and one deciphering), breaking the claimed vil-stprp, stprp as well as the basic sprp security. Our shared difference attack exploits an inherent property of polynomial hash functions called *separability*.
We pinpoint the exact flaw in the security proof of XCB-AES, which arises from the separability of polynomial hash functions. We show that this vulnerability in the XCB design strategy has gone unnoticed for over 20 years and has been inadvertently replicated in many XCB-style TEM designs, including the IEEE 1619.2 standard XCB-AES. We also apply the shared difference attack to other TEMs based on XCB – XCBv1, HCI, and MXCB, invalidating all of their security claims, and discuss some immediate countermeasures.
Our findings are the first to highlight the need to reassess the present IEEE 1619.2 standard as well as the security and potential deployments of XCB-style TEMs. [4]

Keywords: Tweakable enciphering mode, disk-sector encryption, IEEE 1619.2, VIL-STPRP, XCBv1, XCB-AES, HCI, shared difference attack

1 Introduction

Enciphering mode, also known as *length-preserving encryption*, is an encryption method whose ciphertext maintains the length of the plaintext and for a random

[4] Our attack was completed and under review since February, 2024. We later responsibly communicated it to IEEE and the authors of XCB [20, 37]. The authors of XCBv2fb [20] confirmed the validity of our attack on 02/09/2024.

but fixed key is a deterministic bijective map. Length-preserving encryption is sought for in systems where data storage and space are critical.

Tweakable enciphering mode (TEM) extends this concept by incorporating an additional public input – tweak, which often includes contextual information such as disk sectors, packet headers, or counters. This tweak enhances flexibility and security against adaptive chosen-plaintext and chosen-ciphertext attacks. TEMs aim to achieve security as variable-input-length strong tweakable pseudorandom permutations (vil-stprp). That is, for an adversary with oracle access to both encryption and decryption and full control over the tweak and input, the TEM should be indistinguishable from a family of independent random permutations indexed by the tweak and input length. A vil-stprp-secure TEM also satisfies security as a strong tweakable pseudorandom permutation (stprp) and a strong pseudorandom permutation (sprp) under any fixed-length inputs.

TEM Applications. TEMs are deployed in disk-sector encryption, key-wrapping, swapfile and file-based encryption (FBE), and packet-based communication protocols that prioritize bandwidth efficiency. TEMs with large fixed input lengths (256 bits or more) are known as wide-block ciphers, offering efficiency advantages for full-disk encryption by processing large data blocks efficiently. If the tweak input of a TEM is guaranteed to be unique for each message – e.g., by using associated data, packet headers, or counters – then a vil-stprp-secure TEM is also an IND-CCA-secure encryption scheme, aligning with the classical security definition for randomized encryption. Furthermore, as shown in [30], an Encode-then-Encipher [9] scheme based on a vil-stprp construction ensures robust security [11] for authenticated encryption.

The importance of secure TEM designs has recently been reinforced by the efforts of NIST (National Institute of Standards and Technology of US) towards TEM standardization with their Accordion mode initiative. In 2023 and 2024 NIST held two workshops whose objectives were stated [39] as: NIST is "*particularly interested in discussing the possibility of standardizing a tweakable wide block encryption technique that could support a large range of input length*" with applications "*such as storage*" and "*security and efficiency of tweakable wide encryption techniques*".

Over the past few decades, the design space of TEMs has expanded significantly, with numerous schemes [18, 19, 22, 23, 27–29, 31, 37, 38, 43] emerging to optimize various trade-offs between security and efficiency. These designs adopt diverse structural approaches, including Encrypt-Mix-Encrypt, Feistel-style, Hash-ECB-Hash, and Hash-CTR-Hash.

The Encrypt-Mix-Encrypt strategy was first introduced in EME [29] and later refined in AEZ-core [31]. The Hash-ECB-Hash technique has been employed in TET [28], HEH [43], and CTET+ [22], while the Hash-CTR-Hash approach is used by TEMs such as XCBv1 [37], XCBv2 [38], FAST [19], HCTR2 [23], and MXCB [40].

Among these, the Hash-CTR-Hash approach has garnered significant attention in both research and practice due to its efficiency optimizations, leveraging

XOR-universal hashing and parallelizable counter-mode encryption with hardware acceleration support on common platforms.

The IEEE 1619.2 TEM Standard. XCB-AES (originally known as XCBv2 [38]) and EME2-AES [27] are the two TEMs included in the IEEE 1619.2 standard for encryption of shared storage media [32]. These modes were originally standardized in 2010 and were also retained in the 2021 revision of the standard.

XCB-based variants, including XCBv1, XCB-AES, and MXCB, achieve some of the best performances in literature while offering provable claims of $n/2$-bit stprp security. However, despite these guarantees, a crucial question remains:

Are the security foundations of these XCB-based TEMs truly well-understood and solid?

In this work, we answer this question in the negative. We show that XCB-style TEMs are built on fragile security foundations, necessitating the reassessment of the present IEEE 1619.2 standard as well as the security analysis and potential deployments of XCB-style TEMs.

Our Contribution. In this work, we critically reassess the security foundations of XCB-style enciphering modes, with a particular focus on XCB-AES as standardized in IEEE 1619.2 [32]. While XCB-AES has been supported by formal security proofs for block-aligned messages, we present a strikingly simple plaintext recovery attack (named the *shared difference attack*) which fundamentally undermines its security guarantees.

Our attack is highly efficient and requires only two adversarial queries (one enciphering and one deciphering) to recover nearly the entire plaintext, thereby invalidating XCB-AES's previously established claims of sprp, stprp, and vil-stprp security.

To clarify the hierarchy of target security for TEMs and the implication strength of our attack, we also introduce a unified notion for enciphering schemes that includes popular TEM notions as special cases.

We further investigate the root cause of this vulnerability, identifying the separability property of polynomial hash functions that contradicts the unpredictability of counter mode IVs used internally in XCB-AES – a critical flaw in XCB-AES's security proofs and an oversight that has persisted undetected for over two decades. This insight allows us to extend our attack to other TEMs, including HCI [40] and MXCB [40], both of which exhibit similar weaknesses.

Moreover, by leveraging a fundamental provable security result, we demonstrate that even XCBv1 (the original and supposedly stronger version of XCB-AES with independent hash keys) is vulnerable. Our attack breaks its sprp, stprp, and vil-stprp security in just four queries, proving that all XCB-style designs suffer from the same fundamental flaw.

To highlight additional vulnerabilities in the involution property of HCI, we introduce a different three-query distinguishing attack (named the *flipped parts attack*) which compromises its security as a pseudorandom involution (later

formalized as prs[5] in Section 3). Notably, this attack remains effective even if the underlying hash functions of HCI are inseparable or replaced with stronger primitives such as pseudorandom functions (PRFs).

By invalidating these security claims, our findings necessitate a re-evaluation of the IEEE 1619.2 standard and raise alarm for real-world deployments of XCB-style TEMs. We also discuss immediate countermeasures to mitigate these vulnerabilities within the Hash-CTR-Hash TEM paradigm.

Our work not only exposes a crucial flaw in a long-standing encryption standard, but also illustrates the fragility of assumedly "well-understood" cryptographic design principles relying on separability in polynomial hashing.

Attack Overview: The shared difference attack is a plaintext recovery attack under a chosen-ciphertext attack (CCA) model. For an m-bit ciphertext, the attack recovers the first $m - n$ bits of the corresponding plaintext. Here, m can be quite large, such as a typical sector size in disk encryption (e.g., 512B or 4KiB), and n represents the block size of the encryption primitive (16B for AES-128 in this example). Consequently, the attack is capable of recovering almost the entire sector.

Informally, the attack proceeds as follows: Given a target ciphertext C, the attacker performs two steps: 1. The attacker decrypts a related ciphertext $C' = C \oplus \Delta$, where Δ is an arbitrary non-zero binary string of length $|C|$, with the last n bits set to 0. This results in a new message, M'. 2. The attacker then encrypts $M'' = M' \oplus \Delta$, producing the ciphertext C''. Finally, the plaintext is recovered using the formula $M = C \oplus M'' \oplus C''$, where the first $m - n$ bits of M matches the first $m - n$ bits of the target plaintext.

Related Work. Our shared difference attack is the first plaintext recovery attack on XCB-AES, HCI, and MXCB. It requires only two queries – one enciphering and one deciphering – to recover all but the last block of plaintext for any given ciphertext. In doing so, it completely breaks the basic sprp security of XCB-AES and MXCB, as well as the basic prs security of HCI.

Basing their result on knowledge of our shared-difference attack, the follow-up work by Wang et al. [46] demonstrates how our attack approach can be used to also recover the last plaintext block at the cost of five additional TEM queries.

Their attack relies on varying tweaks and invalidates only the stprp security of XCBv1 and XCB-AES. In contrast, our shared difference attack keeps the tweak fixed and therefore, breaks even the basic sprp security of all XCB variants.

sprp attacks are stronger than stprp attacks as they also work with fixed, zero, or empty tweaks.[6]

All these findings expose fundamental security flaws in XCB-style TEMs. Our work serves as the foundation for these discoveries by introducing hash

[5] PRI is a common abbreviation used for pseudorandom injection. Therefore, we use PRS; short for pseudorandom self-inverse permutation

[6] stprp attacks are more limited in practice as some TEM applications do not support tweaks such as – key wrapping, wide block ciphers, deterministic encryption for encrypted database/keyword search, etc.

separability, identifying the security vulnerabilities it introduces, and presenting the first and most efficient attack that exploits this weakness to break the core security guarantees of these schemes.

Paper Organization. The paper is structured as follows. Section 2 covers preliminaries, including notation and cryptographic definitions. Section 3 introduces TEM security notions such as vil-stprp. Section 4 describes the XCB-AES specification and its existing security claims. Section 5 presents our shared difference attack, breaking XCB-AES in just two queries. Section 6 extends the attack to other XCB-style TEMs – XCBv1, HCI and MXCB and explores potential countermeasures. Finally, Section 7 summarizes key findings, proposed defenses, and open problems.

2 Preliminaries

2.1 Notation

Strings. All strings used here are binary strings. Strings of length $n > 0$ are referred to as n-bit strings, and the set of all n-bit strings is denoted by $\{0,1\}^n$. The set of strings of any possible length is denoted as $\{0,1\}^*$. The set of all permutations of $\{0,1\}^n$ is denoted as $\mathrm{Perm}(n)$ and the set of all self-inverse permutations or involutions of $\{0,1\}^n$, is denoted by $\mathrm{SIPerm}(n)$, where $\mathrm{SIPerm}(n) \subset \mathrm{Perm}(n)$. For any string A, $|A|$ is the length of A in bits. For two strings $A, B \in \{0,1\}^*$ with (w.l.o.g.) $|A| \leq |B|$, we let $A \oplus B$ denote the bitwise XOR of $A \| 0^{|B|-|A|}$ and B, and define $A \oplus_a B = (A \oplus B)[0 \dots a-1]$. We use $\mathsf{msb}_a(A)$ to denote the prefix string $A[0 \dots a-1]$. For two a-bit strings X and Y, we denote their field multiplication in $\mathrm{GF}(2^a)$ (using a suitable, fixed irreducible polynomial) as $X \cdot Y$.

Partitions. Given a string A and an integer n such that $|A| = cn + d$, where c is a positive integer and $0 < d \leq n$, the notation $A_1, A_2, \dots, A_{c+1} \xleftarrow{n} A$ is used to indicate the partitioning of A into a maximum number of n-bit blocks. Each block A_i has a length of n for $1 \leq i \leq c$, and the last block A_{c+1} has a length of d. For a given n (as used in partitioning), we define the zero-padding function for A as $\mathsf{pad}_n(A) = A_1 \| A_2 \| \dots \| A_{c+1} \| 0^{n-d}$. We define a length-appended injective padding as $\mathsf{pad2}_n(A) = \mathsf{pad}_n(A) \| a$ where a is n-bit binary representation of $|A|$.

Adversary Interaction. An adversary $\mathcal{A}$ is a probabilistic algorithm that interacts with oracles and produces an output of either 0 or 1. Oracles are represented as superscripts. The notation $\mathcal{A}^{\mathcal{O}_1, \mathcal{O}_2} \Rightarrow 1$ signifies the event where the adversary $\mathcal{A}$ queries the oracles $\mathcal{O}_1$ and $\mathcal{O}_2$ before eventually outputting the bit 1.

Miscellaneous. We let $\mathbb{N}$ denote the set of natural numbers. The notation $r \leftarrow^{\$} \mathcal{R}$ indicates the random sampling of an element r from a finite set $\mathcal{R}$ with a uniform distribution. For a positive integer z, we use $\mathsf{bin}_n(z)$ to denote the n-bit binary representation of z.

2.2 Definitions

In this section, we provide the syntax of a TEM followed by the standard definitions of relevant cryptographic primitives in this work. The security definitions for TEMs are provided in Section 3.

Definition 1 (Tweakable Enciphering Mode (TEM)). *A TEM scheme is defined as* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *which consists of three components: a key distribution* $\mathcal{K}$*, a deterministic encryption algorithm* $\mathcal{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{C}$ *that preserves the length of the message, and a deterministic decryption algorithm* $\mathcal{D} : \mathcal{K} \times \mathcal{T} \times \mathcal{C} \to \mathcal{M}$*. The encryption algorithm takes a key* K*, a tweak* T*, and a message* M *from the sets* $(K, T, M) \in \mathcal{K} \times \mathcal{T} \times \mathcal{M}$*, and returns a ciphertext* $C = \mathcal{E}(K, T, M) = \mathcal{E}_K(T, M)$*. The decryption algorithm maps the key, tweak, and ciphertext to the original message using* $M = \mathcal{D}(K, T, C) = \mathcal{D}_K(T, C)$*. Here being length-preserving means that for any given key* K*, tweak* T*, and message* M*,* $|\mathcal{E}_K(T, M)| = |M|$*.*

Definition 2 (Block Cipher). *A block cipher* $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ *is a function that is a permutation for every* $K \in \{0,1\}^k$*. We use* $E_K(\cdot)$ *to represent* $E(K, \cdot)$ *for every* K*. A block cipher can be seen as a TEM with tweak fixed to some constant,* $\mathcal{K} = \{0,1\}^k$ *and* $\mathcal{M} = \mathcal{C} = \{0,1\}^n$*.*

Definition 3 (Universal Hashing). *Let* $H : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^m$ *for some non-negative* k *and* m *be a set of keyed hash functions. Let* H_K *denote* H *with a* k*-bit key* K*.* H *is called* ϵ*-XOR-universal if for any given two distinct inputs* $X, X' \in \{0,1\}^*$ *and any output* $Y \in \{0,1\}^m$*,*

$$\Pr[H_K(X) \oplus H_K(X') = Y] \leq \epsilon,$$

where the probability is computed over K *chosen uniformly at random from* $\{0,1\}^k$*. Further, when* Y *is fixed to* 0^m*,* H *is called* ϵ*-universal.*

Definition 4 (Polynomial Hashing). *A polynomial hash* $H : \{0,1\}^m \times \{0,1\}^* \to \{0,1\}^m$ *for some non-negative* m *is a set of keyed hash functions. For any given inputs* $(K, X) \in \{0,1\}^m \times \{0,1\}^*$ *with* $P_1, P_2, \ldots, P_c \xleftarrow{m} \mathsf{pad2}_m(X)$ *for some positive* c*, we define two polynomial hashes* G *and* H *as follows:*

$$\mathsf{G}(K, X) = \mathsf{G}_K(X) = K^{c-1} \cdot P_1 \oplus K^{c-2} \cdot P_2 \oplus \ldots \oplus K \cdot P_{c-1} \oplus P_c\,,$$
$$\mathsf{H}(K, X) = \mathsf{H}_K(X) = K \cdot \mathsf{G}_K(X)\,.$$

G and H are $\ell \cdot 2^{-m}$-universal and $\ell \cdot 2^{-m}$-XOR-universal hash functions, respectively when restricted to messages with post-padding length $m \cdot \ell$, i.e. setting $c = \ell$ in Definition 4.

3 Security Notions for Enciphering Schemes

In this section, we introduce a unified notion for enciphering schemes which incorporates popular TEM notions as its special cases.

Let $\Pi = (\mathcal{K}, \mathsf{TEM}, \mathsf{TEM}^{-1})$ be a TEM scheme with key space $\mathcal{K}$, tweak space $\mathcal{T}$ and message space $\mathcal{M} = \cup_{i\in\mathcal{L}}\{0,1\}^i$ for some set $\mathcal{L}$ of all allowed message lengths. Let $\mathcal{T}^\star = \mathcal{T}$ when $|\mathcal{T}| \neq 0$ and $\mathcal{T}^\star = \{0\}$, otherwise. Consider an adversary $\mathcal{A}$ whose goal for a given $\alpha \in \{0,1\}$ is to distinguish between the functions $(\mathsf{TEM}_K, \alpha\mathsf{TEM}_K^{-1})$ with $K \leftarrow^{\$} \mathcal{K}$ and a randomly sampled permutation family $(\pi_{\mathcal{T},\mathcal{L}}, \alpha\pi_{\mathcal{T},\mathcal{L}}^{-1})$ by making oracle queries to them where

$$(\pi_{\mathcal{T},\mathcal{L}}, \pi_{\mathcal{T},\mathcal{L}}^{-1}) = \cup_{i\in\mathcal{L}}\{\{(\pi_{T,i}, \pi_{T,i}^{-1})\}_{T\in\mathcal{T}^\star} \subseteq \mathcal{F}_i\}$$

i.e. for each distinct tweak $T \in \mathcal{T}^\star$ or message length $i \in \mathcal{L}$, the corresponding permutation $\pi_{T,i}$ is uniformly sampled at random from some set $\mathcal{F}_i \subseteq \mathrm{Perm}(i)$ with $\pi_{T,i}^{-1}$ defining the inverse map of $\pi_{T,i}$. Here the oracle $\alpha\pi_{T,i}^{-1}(\cdot,\cdot)$ returns the output of $\pi_{T,i}^{-1}$ when $\alpha = 1$; otherwise, it returns an empty string. In other words, α determines whether the adversary has access to the inverse oracle or not. The advantage of $\mathcal{A}$ in breaking the type-security of Π is defined as follows:

$$\begin{aligned}\mathbf{Adv}_{\Pi}^{\mathsf{type}}(\mathcal{A}) = \big|\Pr[K \leftarrow^{\$} \mathcal{K} : \mathcal{A}^{\mathsf{TEM}_K(\cdot,\cdot),\alpha\mathsf{TEM}_K^{-1}(\cdot,\cdot)} \Rightarrow 1] \\ - \Pr[\cup_{i\in\mathcal{L}}\{(\pi_{\mathcal{T},i}, \pi_{\mathcal{T},i}^{-1}) \subseteq \mathcal{F}_i\} : \mathcal{A}^{\pi_{\mathcal{T},\mathcal{L}}(\cdot,\cdot),\alpha\pi_{\mathcal{T},\mathcal{L}}^{-1}(\cdot,\cdot)} \Rightarrow 1]\big|,\end{aligned}$$

where $\pi_{\mathcal{T},\mathcal{L}}(T,M) = \pi_{T,|M|}(M)$ and $\pi_{\mathcal{T},\mathcal{L}}^{-1}(T,C) = \pi_{T,|C|}^{-1}(C)$ for every T, M and C.

Now that the generic definition of type is defined, we recall all the existing TEM security notions of (variable-input-length) (strong) (tweakable) pseudorandom permutation ((vil-)(s)(t)prp) and (variable-input-length) (tweakable) pseudorandom self-inverse permutation ((vil-)(t)prs) as special cases of it in Table 1.

type	prp	sprp	tprp	stprp	vil-prp	vil-sprp	vil-tprp	vil-stprp	prs	tprs	vil-prs	vil-tprs
α	0	1	0	1	0	1	0	1	0			
$\lvert\mathcal{T}\rvert$	≤ 1		≥ 2		≤ 1		≥ 2		≤ 1	≥ 2	≤ 1	≥ 2
$\lvert\mathcal{L}\rvert$	1				≥ 2				1		≥ 2	
$\mathcal{F}_i$	$\mathrm{Perm}(i)$								$\mathrm{SIPerm}(i)$			

Table 1: Comparison of various TEM or wide block enciphering security notions. Here α is an indicator function that when 1 allows the inverse oracle access to the adversary and restricts otherwise. $|\mathcal{T}| \leq 1$ means that the targeted TEM is not tweakable or the tweak is fixed for the adversary. $|\mathcal{L}| = 1$ means that the targeted TEM is a fixed-input-length function and variable-input-length, otherwise.

We emphasize that vil-stprp is the strongest TEM security notion among the others in Table 1 and is the primary target in many applications and security analyses due to its practical security advantages. For a detailed discussion on the practical applications of vil-stprp schemes, see Appendix A.

4 XCB-AES

XCB (eXtended Code Book) is a popular TEM that was proposed by McGrew and Fluhrer in 2004 [37]. Initially, XCB lacked a formal security proof. In 2007 [38], the authors revised XCB (from XCBv1 to XCBv2) and provided a proof for their updated version. In 2013, Chakraborty et al. [20] analyzed both versions of XCB and demonstrated that the security claims for XCBv2 were flawed, particularly for messages that do not align with the block length of the block cipher. They presented a distinguishing attack to illustrate the identified vulnerability. In the same paper, they provided a new proof for XCBv1 and XCBv2, but only for full-block messages. They named this variant of XCBv2 XCBv2fb. XCBv2, also known as XCB-AES, is included in the IEEE 1619.2 standard [32] for encryption of sector-oriented storage media. In the following section, we formally define the XCB-AES TEM as per the standard specification.

4.1 XCB-AES: Specification

The XCB-AES algorithm is based on a block cipher E (instantiated with AES), two hash functions H_1 and H_2 and the counter mode of encryption (as a one-time pad generator) G as depicted in Figure 1. To understand XCB-AES, let us first define the subcomponents of XCB-AES.

Hash Functions H_1 and H_2. Let $H_1 : \{0,1\}^k \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^n$ and $H_2 : \{0,1\}^k \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^n$ be two keyed hash functions defined in terms of the polynomial hash H (see Definition 4) as:

$$\begin{aligned} H_1(L, A, B) &= \mathsf{H}_L(0^n \| \mathsf{pad}_n(A) \| \mathsf{pad}_n(B) \| 0^n)\,, \\ H_2(L, A, B) &= \mathsf{H}_L(\mathsf{pad}_n(A) \| 0^n \| \mathsf{pad}_n(B) \| \mathsf{len}_{A,B}) \end{aligned}$$

where $\mathsf{len}_{A,B} = \mathsf{bin}_{n/2}(|A\|0^n|) \| \mathsf{bin}_{n/2}(|B|)$.

Counter Mode G. Let $G[E] : \{0,1\}^k \times \{0,1\}^n \times \mathbb{N} \to \{0,1\}^*$ be the counter mode of encryption that uses a block cipher E as an underlying primitive and is defined in the form of a one-time-pad generator as:

$$\begin{aligned} G(L, IV, \ell) = G_{L,\ell}(IV) = &(E_L(IV) \| E_L(IV \oplus \mathsf{bin}_n(1)) \| \cdots \\ &\cdots \| E_L(IV \oplus \mathsf{bin}_n(\lceil \ell/n \rceil - 1)))[0 \ldots \ell - 1]\,. \end{aligned}$$

We can now formally give the definition of XCB-AES as specified in the IEEE 1619.2-2021 standard, which consists of two algorithms: encipher and decipher. The encipher algorithm takes a key $K \in \{0,1\}^k$, a tweak $T \in \{0,1\}^t$ and a message $M \in \{0,1\}^m$ for some positive integers $k \leq 2n$, t and m where $n = 128$ bits. We split M and denote it by $M_L \| M_R$ with $|M_R| = n$.

Key Derivation. XCB-AES uses its main key K to derive an n-bit key K_h and three k-bit keys K_e, K_d and K_c as:

$$\begin{aligned} K_h &= E_K(\mathsf{bin}_n(0)), \\ K_e &= \mathsf{msb}_k(E_K(\mathsf{bin}_n(1))\|E_K(\mathsf{bin}_n(2))), \\ K_d &= \mathsf{msb}_k(E_K(\mathsf{bin}_n(3))\|E_K(\mathsf{bin}_n(4))), \\ K_c &= \mathsf{msb}_k(E_K(\mathsf{bin}_n(5))\|E_K(\mathsf{bin}_n(6))) \,. \end{aligned}$$

It then uses two keyed hash functions H_1 and H_2 with key K_h, the counter mode G with key K_c and two keyed instances of a block cipher E as $E_{K_e} : \{0,1\}^n \to \{0,1\}^n$ and $E_{K_d} : \{0,1\}^n \to \{0,1\}^n$ (with inverse calls denoted as $E_{K_e}^{-1}$ and $E_{K_d}^{-1}$) to compute:

$$C_L = G_{K_c,m-n}(E_{K_e}(M_R) \oplus H_1(K_h, T, M_L)) \oplus M_L \,, \tag{1}$$

$$C_R = E_{K_d}^{-1}(E_{K_e}(M_R) \oplus H_1(K_h, T, M_L) \oplus H_2(K_h, T, C_L)) \tag{2}$$

returns C as $C_L\|C_R$. For ease of understanding, a pictorial diagram for XCB-AES enciphering is provided in Figure 1.

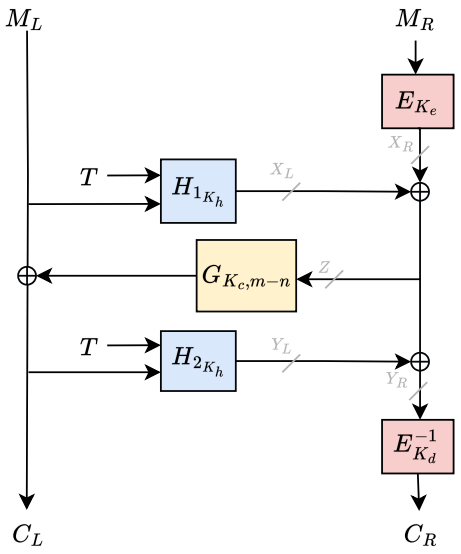


Fig. 1: XCB-AES Block Diagram.

The XCB-AES decipher algorithm swaps the block cipher keys and hash functions. For the same keys and ciphertext C, it is defined as:

$$M_L = G_{K_c,m-n}(E_{K_d}(C_R) \oplus H_2(K_h, T, C_L)) \oplus C_L \,, \tag{3}$$

$$M_R = E_{K_e}^{-1}(E_{K_d}(C_R) \oplus H_2(K_h, T, C_L) \oplus H_1(K_h, T, M_L)) \tag{4}$$

and returns the plaintext $M_L \| M_R$.

Remark. We note that the XCB-AES specification also sets $t \leq n$ and $m \leq 2^{39}$ but the attack results presented in this paper are generic and apply to unbounded t and m. In particular, our attacks exploit a crucial property of polynomial hash functions called *separability* that has remained unidentified in all existing analyses and applies to XCB-AES regardless of whether or not the messages are full-block. In other words, our XCB-AES attack in its generality applies to the latest version of XCB-AES, namely XCBv2fb [20].

4.2 XCB-AES: Existing Security Claim

Currently, XCB-AES has one standing provable security result in [20] for full-block messages. We summarize this result in Theorem 1.

Theorem 1. *Let $\Pi[E]$ be the* XCB-AES *TEM with E as the underlying block cipher then for any adversary $\mathcal{A}$ who makes at most q_e enciphering and q_d deciphering queries to Π with inputs (i.e., tweak and message combined) of length multiple of n and upper bounded by ℓn bits, we have*

$$\mathbf{Adv}^{\mathsf{vil\text{-}stprp}}_{\Pi[E]}(\mathcal{A}) \leq \frac{(5+2^{22})\ell q \sigma}{2^n}\,.$$

where $q = q_e + q_d$ and σ is the total number of n-bit blocks processed within these q queries.

In simple terms, this result says that XCB-AES is a vil-stprp up to at least $q = 2^{(n-23)/3}$ queries and s(t)prp up to $q = 2^{(n-23-\log_2(l))/2}$ queries where l is the length of the inputs in n-bit blocks. Since for AES, the block size is $n = 128$, we can deduce from this result that XCB-AES should be secure as a vil-stprp, stprp and sprp up to 2^{35}, $2^{52-\log_2(\sqrt{l})}$ and $2^{52-\log_2(\sqrt{l})}$ queries, respectively.

Our generic sprp attack of only *two queries* for inputs $> n$ bits invalidates all the latter security claims.

Remark. Our attack does not apply to XCB-AES when the input length is limited to *only* n bits. However, it is easy to see that for n-bit messages, XCB-AES and XCBv1 reduces to LRW1 [35] which is already prone to the basic stprp attack originally proposed by Khairallah in [34] (this simple stprp insecurity of LRW1 was known since [35] but the first formal description of the attack was given in [34]; see their page 5-6, "Tightness of LRW1").

5 Breaking XCB-AES

In this section, we show that for full-block messages of length $> n$ bits, XCB-AES is not sprp secure, which implies that stprp and vil-stprp insecurity

also follows. We present a plaintext recovery attack with only two queries, one forward and one inverse call to XCB-AES.

Core Idea of the Attack. Let us recall from Section 4.1 that both hash functions used in XCB-AES (i.e., H_1 and H_2) are based on the polynomial hash H (Def. 4) which are XOR-universal hash functions.

Furthermore, recall from the enciphering and deciphering algorithms of XCB-AES that H_1 and H_2 are always XORed before evaluating the final E^{-1} call to return the final rightmost output block i.e, C_R in an enciphering call and M_R in a deciphering call (Eq. 2 and 4).

Since H is a keyed XOR-universal hash function, it appears natural to assume that the sum of two independent instances of H (where independence is achieved with either independent random keys or with independent random inputs) remains XOR-universal and, therefore, universal itself. This assumption is also implicitly used in the existing security analysis of XCBv1 [20, Lemma 1], XCBv2 (or XCB-AES) [38, Theorem 1], and XCBv2fb [20, Lemma 1].

Our main observation is that this assumption is *not always valid* and does not hold for all XOR-universal hash functions. Specifically, we identify a crucial structural property of polynomial hash functions, such as H used in XCB-style TEMs, that contradicts the assumption. Our finding directly invalidates the security bounds of XCBv2 (or XCB-AES) and XCBv2fb.

Furthermore, as we will later demonstrate in Section 6.1, this observation extends trivially to XCBv1 and invalidates its security bound as well.

We call this property *separability* and define it as follows:

Definition 5 (Hash Functions Separability). *Let $f : \{0,1\}^k \times \{0,1\}^{kl} \to \{0,1\}^k$ be a function that takes a k-bit key K and an l-block message $M = M_1 \| \ldots \| M_l$ as inputs with each block of size k bits, and returns a k-bit output h. We say that f is separable if for any two l-block messages A and B,*

$$f(K, A \oplus B) = f(K, A) \oplus f(K, B)\,.$$

One can easily notice that the polynomial hash H is separable because

$$\begin{aligned}
&\mathsf{H}(K, A \oplus B) = \\
&\quad = (K^l \cdot (A_1 \oplus B_1) \oplus K^{l-1} \cdot (A_2 \oplus B_2) \oplus \ldots \oplus K \cdot (A_l \oplus B_l)) \\
&\quad = (K^l \cdot A_1 \oplus K^{l-1} \cdot A_2 \oplus \ldots \oplus K \cdot A_l) \oplus (K^l \cdot B_1 \oplus K^{l-1} \cdot B_2 \oplus \ldots \oplus K \cdot B_l) \\
&\quad = \mathsf{H}(K, A) \oplus \mathsf{H}(K, B)\,. \qquad (5)
\end{aligned}$$

We acknowledge that separability aligns with the concept of linearity as discussed in [38, Theorem 2]. However, our contribution lies in identifying and conceptualizing this as a concrete, exploitable property – terming it "separability" – specifically for the way it is used in our attacks.

Before we describe how separability contradicts universality of $H_1 \oplus H_2$ in XCB-AES (which then can be exploited to break XCB-AES), we first give examples of how XOR-universal hash functions are typically used.

XOR-Universal Hash Functions in Practice. XOR-universality is a weaker property compared to collision resistance, offering a different security guarantee. It is particularly relevant in scenarios where the adversary is assumed to have no prior knowledge of the specific hash function instance being used. More formally, the adversary is aware of the entire hash function family, but for each query, a hash function is sampled uniformly at random from this family. This random sampling prevents the adversary from gaining any structural insights into the particular hash function in use beforehand. This relaxed security definition allows for the use of more basic hash constructions, such as polynomial hash functions, to achieve XOR-universality.

For polynomial hash functions, random hash sampling is emulated by resampling the key for each query. A secure XOR-universal hash function is expected to use a freshly sampled key for every query; otherwise, an adversary observing the outputs can exploit its algebraic structure, such as separability, or even recover the key.

However, in practice, polynomial hash function keys are often fixed for efficiency reasons, and security is argued based on post-processing the hash function outputs to prevent adversaries from observing directly the hash function outputs and exploiting any structural weaknesses.

For instance, authenticated encryption (AE) modes like GCM [36] and GCM-SIV [26] reuse the same hash function key across multiple AE calls. However, they avoid security issues by masking each hash function output with a fresh random value, thus preventing the adversary from distinguishing hash function outputs or leveraging structural weaknesses.

XCB-AES follows a similar approach, it post-processes the hash function outputs by making an extra block cipher call. In XCB-AES the unmodified hash function outputs are first XORed (see Eq. 2), which, as we demonstrate, can be exploited to recover the majority of plaintext.

Separability can Contradict Universality. Although H_1 and H_2 differ slightly in design, they are essentially polynomial hash functions with domain-separated inputs. We show that domain separation alone is insufficient to ensure the universality of their sum when used in XCB-AES (or any XCB variant that reuses the same key for both functions).

Consider a k-bit key K, a t-bit tweak T, and two a-bit strings M and C for some positive integers k, t, a. Define the sum of H_1 and H_2 under these inputs as:

$$H_{\mathsf{sum}}(K,T,M,C) = H_1(K,T,M) \oplus H_2(K,T,C) \tag{6}$$

$$= \mathsf{H}_K(0^n \| \mathsf{pad}_n(T) \| \mathsf{pad}_n(M) \| 0^n) \oplus \mathsf{H}_K(\mathsf{pad}_n(T) \| 0^n \| \mathsf{pad}_n(C) \| \mathsf{len}_{T,C})$$

$$= \mathsf{H}_K(\mathsf{pad}_n(T) \| \mathsf{pad}_n(T) \| \mathsf{pad}_n(M \oplus C) \| \mathsf{len}_{T,C}) \oplus K \cdot \mathfrak{L} \tag{7}$$

where $\mathfrak{L} = \mathsf{bin}_n(2n + |\mathsf{pad}_n(T)| + |\mathsf{pad}_n(M)|)$ is the length of H's input (appearing due to the use of input length padding $\mathsf{pad2}$ in H; see Def. 4). We note that $\mathfrak{L}$ is a fixed value for given $|T|$ and $|M| = |C|$, therefore, for any string Δ with

$|\Delta| \leq a$, we have

$$H_{\mathsf{sum}}(K, T, M, C) = H_{\mathsf{sum}}(K, T, M \oplus \Delta, C \oplus \Delta)\,. \tag{8}$$

Thus, even without observing the outputs of H_{sum}, without knowing K, or without choosing T, M, or C, an adversary can generate numerous "trivial" collisions. This evidences to the fact that polynomial hash function outputs need to be preprocessed before XORing in cryptographic applications like XCB-AES, even when the output of their sum is not directly exposed to an adversary.

To establish sprp security, all existing analysis of XCB-AES – XCBv2 [38, Theorem 1], and XCBv2fb [20, Lemma 1] require that the IV in counter mode remains random and unpredictable (and thus unique with high probability) for each TEM query. This requirement is then reduced to the assumption that the probability of a collision in $H_1(K, T, M_i) \oplus H_2(K, T, C_i)$ for some distinct pairs (M_1, C_1) and (M_2, C_2), under the same secret and random key K and tweak T, is sufficiently small.

We show that this assumption is incorrect: regardless of the unknown key K and tweak T, one can construct an arbitrary number of (M_i, C_i) pairs that yield collisions in $H_1(K, T, M_i) \oplus H_2(K, T, C_i)$ with probability 1. This results in a "trivial" IV collision in counter mode within two queries, causing the counter mode to generate the same keystream output twice.

Let A_L denote the all-but-last-block portion of any m-bit string A with $m > n$. Consequently, for a given *target ciphertext* C in XCB-AES, we can construct a chosen ciphertext query C' and a chosen plaintext query M' to force the counter output (denoted as X) – which is used to derive the target ciphertext C – to reappear in our chosen message query M'. By XORing M'_L with the all-but-last blocks of its ciphertext, we can recover X. As a result, all but the last block of the plaintext underlying C can be reconstructed as $C_L \oplus X$, achieving near-complete plaintext recovery with just two XCB-AES queries.

We call this attack – *the shared difference attack* – as each one of the two chosen queries are constructed by XORing the same difference Δ to the previous output. We give a formal definition below.

5.1 The Shared Difference Attack: XCB-AES Plaintext Recovery

The shared difference attack against XCB-AES requires two queries – one enciphering and one deciphering. Let $\mathcal{A}$ be an adversary against XCB-AES who aims to recover *all-but-last* blocks of the plaintext $M \in \{0,1\}^m$ for a given ciphertext $C \in \{0,1\}^m$ and tweak $T \in \{0,1\}^t$ where $m > n$ and $t \geq 0$ are some integers. $\mathcal{A}$ follows the steps from Figure 2 to construct M_L. $\mathcal{A}$ returns the recovered left $(m-n)$ bits of the target plaintext M as M_L.

Attack Correctness. For the target ciphertext $C = C_L \| C_R$ and its corresponding secret plaintext $M = M_L \| M_R$, we have the following relations from Eq. 3 and 4:

A. Setup:

1. Split C as $C_L \| C_R$ where $|C_R| = n$
2. Choose a non-zero constant $\Delta \in \{0,1\}^{m-n}$

B. XCB-AES Oracle Queries:

3. Query $(C_L \oplus \Delta) \| C_R$ with a tweak T to the deciphering oracle and obtain $M'_L \| M'_R$ where $|M'_R| = n$
4. Query $(M'_L \oplus \Delta) \| M'_R$ with a tweak T to the enciphering oracle and obtain $C'_L \| C'_R$ where $|C'_R| = n$

C. Message Recovery:

5. Set M_L to $(C_L \oplus M'_L \oplus C'_L \oplus \Delta)$

Fig. 2: The Shared Difference Attack against XCB-AES.

$$\begin{aligned} M_L \oplus C_L &= G_{K_c,m-n}(E_{K_d}(C_R) \oplus H_2(K_h, T, C_L))\,, \\ E_{K_e}(M_R) \oplus E_{K_d}(C_R) &= H_{\mathsf{sum}}(K_h, T, M_L, C_L)\,. \end{aligned} \tag{9}$$

Similarly, using Eq. 3 and 4 over the input and output of the deciphering query made by $\mathcal{A}$, we get the following relations:

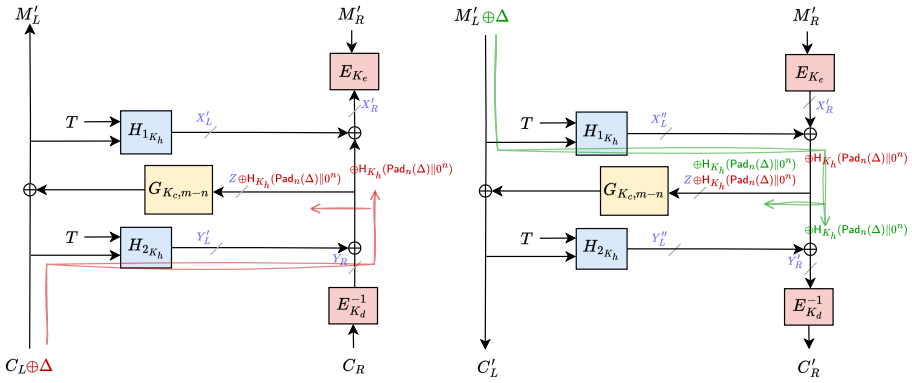
$$\begin{aligned} M'_L \oplus (C_L \oplus \Delta) &= G_{K_c,m-n}(E_{K_d}(C_R) \oplus H_2(K_h, T, C_L \oplus \Delta))\,, \\ E_{K_e}(M'_R) \oplus E_{K_d}(C_R) &= H_{\mathsf{sum}}(K_h, T, M'_L, C_L \oplus \Delta)\,. \end{aligned} \tag{10}$$

Finally, from the input and output of the enciphering query made by $\mathcal{A}$ and Eq. 1 and 2, we get:

$$\begin{aligned} (M'_L \oplus \Delta) \oplus C'_L &= G_{K_c,m-n}(E_{K_e}(M'_R) \oplus H_1(K_h, T, M'_L \oplus \Delta))\,, \\ E_{K_e}(M'_R) \oplus E_{K_d}(C'_R) &= H_{\mathsf{sum}}(K_h, T, M'_L \oplus \Delta, C'_L)\,. \end{aligned} \tag{11}$$

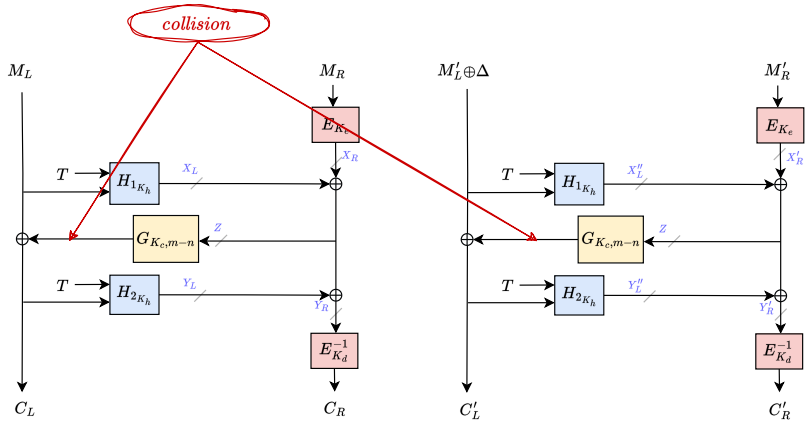
Let us now combine Eq. 9 and 11 together to obtain,

$$\begin{aligned} &(M'_L \oplus \Delta) \oplus C'_L \oplus M_L \oplus C_L \oplus G_{K_c,m-n}(E_{K_d}(C_R) \oplus H_2(K_h, T, C_L)) \\ &\quad = G_{K_c,m-n}(E_{K_e}(M'_R) \oplus H_1(K_h, T, M'_L \oplus \Delta)) \\ &\quad = G_{K_c,m-n}(E_{K_e}(M'_R) \oplus H_2(K_h, T, C_L) \oplus H_{\mathsf{sum}}(K_h, T, M'_L \oplus \Delta, C_L)) \quad (12) \\ &\quad = G_{K_c,m-n}(E_{K_e}(M'_R) \oplus H_2(K_h, T, C_L) \oplus H_{\mathsf{sum}}(K_h, T, M'_L, C_L \oplus \Delta)) \quad (13) \\ &\quad = G_{K_c,m-n}(E_{K_d}(C_R) \oplus H_2(K_h, T, C_L))\,. \quad (14) \end{aligned}$$



Query 1: Deciphering $(C_L \oplus \Delta)\|C_R$ Query 2: Enciphering $(M'_L \oplus \Delta)\|M'_R$

Fig. 3: Block Diagram of Shared Difference Queries to XCB-AES.



Target Ciphertext: $C = C_L\|C_R$ Query 2: Enciphering $(M'_L \oplus \Delta)\|M'_R$

Fig. 4: Block Diagram Showing Collision that Results in Recovery of M_L.

Here Eq. 12 holds from the definition of H_{sum}, Eq. 13 holds from Eq. 8 by adding Δ to both $(M'_L \oplus \Delta)$ and C_L and Eq. 14 holds from Eq. 10. Simplifying the expression gives us $M_L = C_L \oplus M'_L \oplus C'_L \oplus \Delta$ i.e., $\mathcal{A}$ successfully recovers M_L (which is all but last n bits of M) within two oracle queries and with probability 1.

A pictorial diagram with simplified attack description is provided in Figures 3 and 4. Here, Xs, Ys, and Z are internal variables, as shown in Figure 1. The colored sketch arrows illustrate how the Δ difference propagates through the TEM calls, ultimately canceling out in G's inputs across two TEM queries, leading to a collision in corresponding G's outputs.

6 Shared Difference Attack Beyond XCB-AES

6.1 Application to XCBv1

XCBv1 vs. XCB-AES. XCBv1 [37] is the predecessor of XCB-AES and represents a *stronger* variant of XCB. It employs *independent keys* in its internal hash functions.

Specifically, XCBv1 differs from XCB-AES in the following ways:

1. XCBv1 (illustrated in Figure 5) can be viewed as a vertically mirrored version of XCB-AES, with the counter component shifted to the right, and the block ciphers shifted to the left. Consequently, XCBv1 defines the left-right partitioning of inputs in reverse, i.e., for any $M = M_L \| M_R$, we now have $|M_L| = n$.
2. XCBv1 restricts the key length k to n and derives the keys as follows:

$$\begin{aligned} K_e &= E_K(\mathsf{bin}_n(0)), \\ K_{h1} &= E_K(\mathsf{bin}_n(1)), \\ K_c &= E_K(\mathsf{bin}_n(2)), \\ K_{h2} &= E_K(\mathsf{bin}_n(3)), \\ K_d &= E_K(\mathsf{bin}_n(4)) \end{aligned}$$

   where K_{h1} and K_{h2} serve as hash function keys for the first and second internal hash functions of the XCBv1 enciphering call, respectively.
3. XCBv1 employs the same hash function (let us denote it by H) but with different keys to replace the H_1 and H_2 calls. For the same inputs L, A and B as considered for H_1 and H_2 in XCB-AES, H is defined as

$$H(L, A, B) = H_L(A, B) = \mathsf{H}_L(\mathsf{pad}_n(A) \| \mathsf{pad}_n(B) \| \mathsf{len}_{A,B}) \,. \tag{15}$$

4. XCBv1 provides tweaks and message inputs to H in reverse order compared to H_1 and H_2 in XCB-AES.

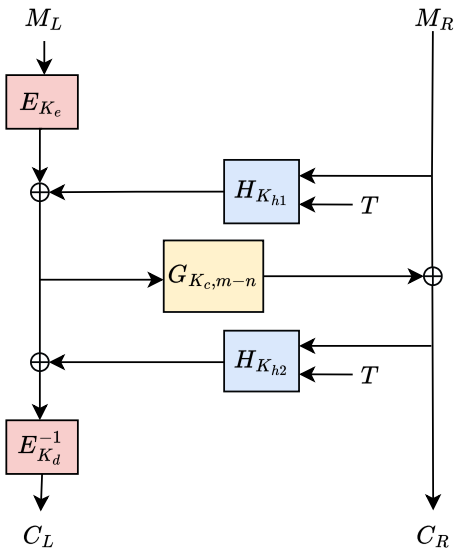


Fig. 5: XCBv1 Block Diagram.

A pictorial representation of XCBv1 is provided in Figure 5. The primary security-relevant distinction is the use of two independent hash keys, while the remaining modifications are not affecting the applicability of the shared difference attack.

Attacking XCBv1. At a first glance, it appears hard to establish a relation like Eq. 8 for $H_1 \oplus H_2$ for XCBv1. We demonstrate that the shared difference attack can still be leveraged against XCBv1 via a proof by contradiction.

Recall that a permutation is called an *involution* if it is its own inverse and the set of all involutions on $\{0,1\}^m$ is denoted by $\mathrm{SIPerm}(m)$, where $\mathrm{SIPerm}(m) \subset \mathrm{Perm}(m)$. Consider the involution $\Pi : \{0,1\}^m \to \{0,1\}^m$ defined as:

$$\Pi(M) = \pi(\pi^{-1}(M) \oplus \gamma)$$

for some fixed non-zero m-bit constant γ and an m-bit permutation π with inverse π^{-1}. An important result from [41, Theorem 1] states:

Theorem 2. *[41, Theorem 1] Let β be an m-bit permutation with cycle c (i.e., β^c is the identity map) and let π be an m-bit random permutation, then the permutation $\pi(\beta(\pi^{-1}(\cdot)))$ is distributed uniformly among the permutations with the same cycle type c as β.*

A simple proof of this theorem is provided in [41]. Setting $\beta(A) := A \oplus \gamma$ yields $c = 1$, making β an involution. This leads to the following corollary:

Corollary 1. *Let γ be some non-zero and fixed m-bit constant and π be a permutation sampled uniformly at random from* $\mathrm{Perm}(m)$*, then the permutation Π defined as $\pi(\pi^{-1}(\cdot) \oplus \gamma)$ is also unifromly distributed among* $\mathrm{SIPerm}(m)$.

Now, assuming XCBv1 is a vil-stprp TEM implies it is also an sprp. We instantiate Π with π as XCBv1 using secret, randomly sampled keys and a fixed tweak. Let F_K denote XCBv1 with tweak $T = 0^n$ and keys $K_e, K_d, K_c, K_{h1}, K_{h2}$, i.e., $K = K_e \| K_d \| K_c \| K_{h1} \| K_{h2}$.

By Corollary 1, the composition $F_K(F_K^{-1}(M) \oplus \gamma)$ is a prs for any nonzero γ. Setting m as a multiple of n and $\gamma = \mathsf{bin}_m(1)$, we analyze this composition using pictorial representations in Figure 6–Figure 8.

Figure 6 shows a pictorial diagram of this composition in terms of F_K and F_K^{-1} i.e., XCBv1 forward and inverse calls with key K and tweak 0^n. Figure 7 simplifies it further by separating the hash function key K_{h1} dependent variables from the rest. These variables appear on the left side of Figure 7 with the form $H_{K_{h1}}(S_R \oplus \mathsf{bin}_{m-n}(1), 0^n) \oplus H_{K_{h1}}(S_R, 0^n)$ which can be simplified using the definition of H as follows:

$$
\begin{aligned}
&H_{K_{h1}}(S_R \oplus \mathsf{bin}_{m-n}(1), 0^n) \oplus H_{K_{h1}}(S_R, 0^n) \\
&= \mathsf{H}_{K_{h1}}((S_R \oplus \mathsf{bin}_{m-n}(1)) \| 0^n \| \mathsf{len}_{S_R, 0^n}) \oplus \mathsf{H}_{K_{h1}}(S_R \| 0^n \| \mathsf{len}_{S_R, 0^n}) \\
&= \mathsf{H}_{K_{h1}}((\mathsf{bin}_{m-n}(1)) \| 0^{2n}) = \mathsf{H}_{K_{h1}}(0^{m-n-1} \| 1 \| 0^{2n}) \\
&= K_{h_1}^3 \, .
\end{aligned}
$$

After restructuring, i.e., splitting $K_{h_1}^3$ to the counter mode branch and the bottom block cipher branch, we obtain Figure 8(a) and further combining the components in red circles, we get to Figure 8(b) where E' (a valid permutation) and G' (a valid function) are modified versions of their counterparts E and G in XCB-AES. Since the shared difference attack is independent of E and G, it can be directly adapted to break the prs security of the composition. This contradicts Corollary 1, proving that XCBv1 is not sprp. This result invalidates the existing security bound of XCBv1 established in [20].

We note that the shared difference attack here closely follows the structure of the one on XCB-AES with minor differences (to adapt the structural changes between XCBv1 and XCB-AES). Hence, we omit the details for the composition $F_K(F_K^{-1}(M) \oplus \gamma)$.

The attack involves two queries to the composition, this translates to four queries to XCBv1 – two enciphering and two deciphering queries. Consequently, breaking the sprp security of XCBv1 using the shared difference attack requires *four queries*.

6.2 Application to HCI and MXCB

HCI vs. XCB-AES. HCI [40] is an XCB-style involution TEM with a proven claim of vil-tprs security. The paper also contains MXCB [40] TEM as an vil-stprp-

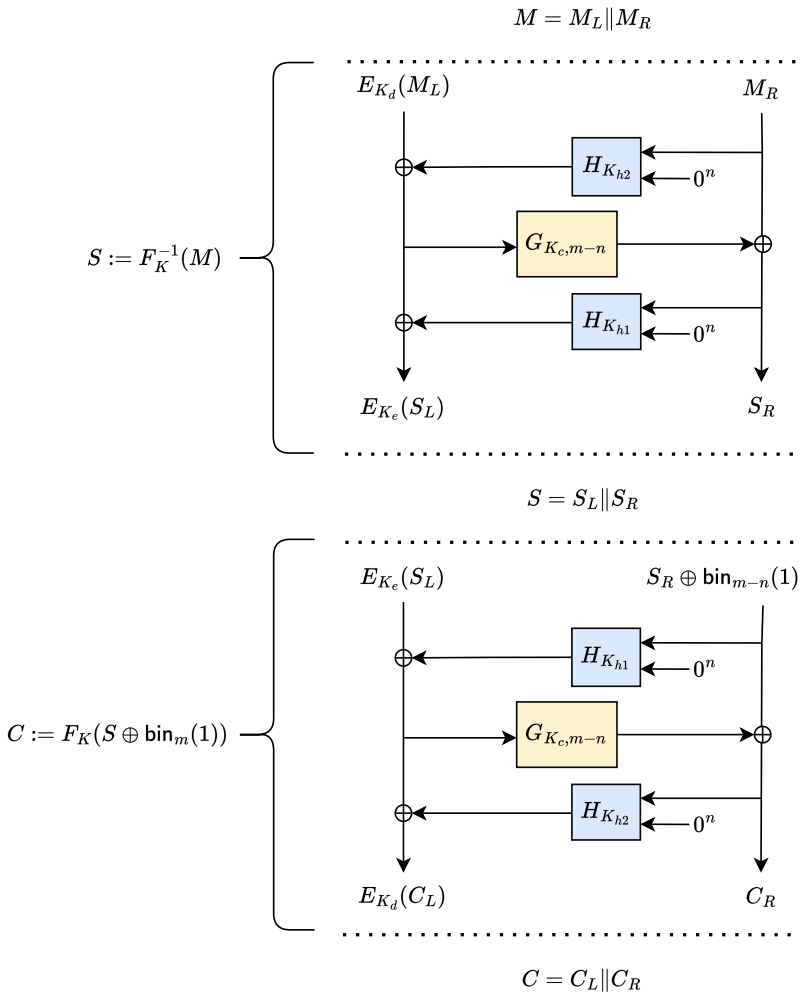


Fig. 6: Pictorial Representation of the Composition $F_K(F_K^{-1}(M) \oplus \gamma)$.

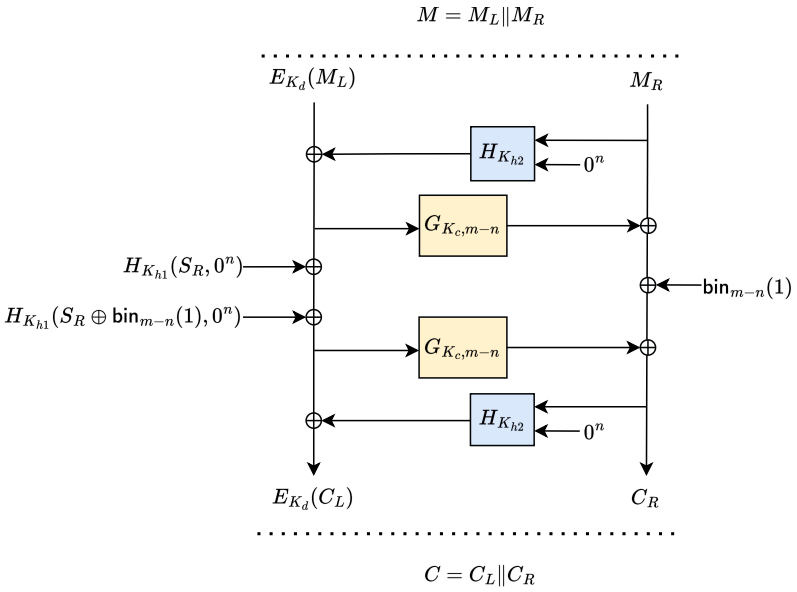


Fig. 7: Simplifying $F_K(F_K^{-1}(M) \oplus \gamma)$: Step 1.

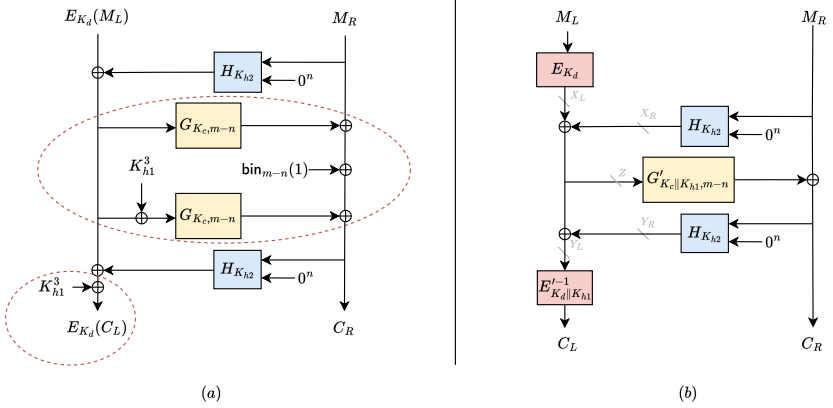


Fig. 8: Simplifying $F_K(F_K^{-1}(M) \oplus \gamma)$: Step 2 (left) and 3 (right).

secure variant of HCI whose security relies on HCI's vil-tprs claim. HCI differs from XCB-AES in the following ways:

1. HCI utilizes the same key for both block cipher calls, or $K_e = K_d$.
2. Instead of using separate hash functions H_1 and H_2, HCI employs a single hash function (denoted as H') with the same key for both hash functions (denoted by K_h). Given the inputs L, A, and B, as applied to XCB-AES, H' is defined as

$$H'(L, A, B) = H'_L(A, B) = \mathsf{H}_L(\mathsf{pad}_n(A) \| \mathsf{pad}_n(B)) .$$

3. Structurally, HCI can be regarded as a vertically mirrored variant of XCB-AES, similar to XCBv1, where the counter component is shifted to the right and the block cipher operations are shifted to the left. As a result, the left-right partitioning of inputs is reversed, such that for any $M = M_L \| M_R$, we have $|M_L| = n$.
4. HCI does not explicitly define how its keys are derived. However, it is mentioned that key derivation can be performed using block cipher calls, similar to the approach used in XCBv1.

We note that, up to insignificant differences, HCI processes the keyed hash functions in the same way as XCB-AES. This allows the shared difference attack to be directly applicable, breaking its vil-tprs security and, consequently, compromising MXCB's vil-stprp security guarantees, thereby invalidating the *all* existing security bounds established in [40]. A block diagram of HCI is provided in Figure 9.

6.3 How to Counter the Shared Difference Attack

As demonstrated, the shared difference attack exploits the separability of the underlying hash functions in XCB variants, leveraging the relation in Eq. 8 when the raw outputs of these separable hash functions are XORed.

Mask the Hash. A working defense is masking the separable hash functions outputs with a random value, freshly sampled for each distinct hash input. This prevents adversaries from exploiting hash function separability. The PIV [44] mode instantiation XCB* [46] is based on this strategy, achieving a provable stprp security bound of $n/2$ bits, albeit with some performance trade-offs. These arise due to the sequential dependency between the first block cipher call and the first hash function, which limits parallelization.

Avoid the Sum. A better countermeasure could be to avoid the hash sum altogether as done in HCTR2 scheme [23]. HCTR2 encrypts the first hash function's outputs using a block cipher call before XORing it to the other hash function's output, thereby disrupting the relation in Eq. 8 and neutralizing the shared difference attack. The HCTR2 scheme [23] offers a stronger security guarantee i.e., vil-stprp security of $n/2$ bits while maintaining the same efficiency as XCB-AES.

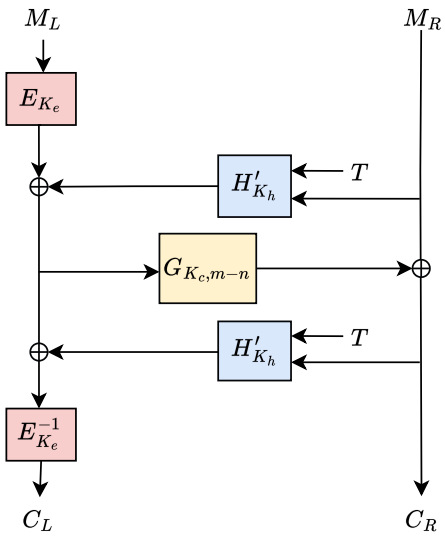


Fig. 9: HCI Block Diagram.

Use Inseparable Hashing. Another better alternative is to simply use inseparable hash functions to invalidate Eq. 8. An example of such a scheme is the Generic Enciphering Mode (GEM) [17]. GEM generalizes both HCTR2 and XCB designs, incorporating variable-input-length pseudorandom functions (VIL-PRFs) for its hashing parts to prevent shared difference attacks and leverages Generic Counter Mode (GCTR) [1] for its CTR part to achieve beyond-birthday-bound (i.e., $> n/2$-bit) security.

To maintain efficiency, GEM supports instantiations with AES and n-to-$8n$ bit AES-based expanding PRF ButterKnife [4]. Fixed-length-expanding symmetric primitives like ButterKnife generalize (T)BCs and their use in GEM shows similar high performance and robust security advantages seen in other applications of expanding primitives. These include: AEAD for short messages [6] and with leakage resilience [12, 24]; robust online AEAD [2, 5, 14]; transciphering-friendly AEAD schemes [16]; pseudorandom number generators [7]; message authentication codes [13] and key derivation functions [15].

6.4 On the Use of Inseparable Hash Functions

Naturally, one might think replacing the polynomial hash function with more robust alternatives, including inseparable XOR-universal functions or even stronger primitives like VIL-PRFs, should counter the attack in general for all XCB-style variants. However, regarding HCI, we now show that even such a replacement with VIL-PRFs fails to retain security. This variant remains sus-

ceptible to another of our distinguishing attacks, which we refer to as the "flipped parts attack".

6.5 The Flipped Parts Attack on HCI

The flipped parts attack targets HCI even in the case when the hash function H'_{K_h} is replaced with a PRF F_{K_h} (over the same domain). The attack requires only three enciphering queries, making it a chosen-plaintext attack (CPA), since enciphering and deciphering operations are identical for involution TEMs.

Let $\mathcal{A}$ be an adversary that wants to distinguish HCI from a random involution using chosen-plaintext queries. $\mathcal{A}$ selects a message $M \in \{0,1\}^m$ and a tweak $T \in \{0,1\}^t$, where $m > n$ and $t \geq 0$ are some integers. The adversary then proceeds with the following steps:

1. Split M into $M_L \| M_R$ such that $|M_L| = n$.
2. Query $(M_L \| M_R)$ with a tweak T to the enciphering oracle and obtain $C_L \| C_R$, where $|C_L| = n$.
3. Query $(M_L \| C_R)$ with a tweak T to the enciphering oracle and obtain $M'_L \| M'_R$, where $|M'_L| = n$.
4. Query $(C_L \| M_R)$ with a tweak T to the enciphering oracle and obtain $C'_L \| C'_R$, where $|C'_L| = n$.

The adversary then checks whether $M_R \oplus C_R = M'_R \oplus C'_R$. If the equality holds, $\mathcal{A}$ concludes that the oracle implements HCI and outputs 1; otherwise, it outputs 0, indicating that the oracle is a random involution.

Attack Correctness. The correctness of the attack follows from the structure of HCI, even when the hash function H'_{K_h} is replaced with a PRF F_{K_h} in Figure 9. Specifically, from all the inputs of G, we derive the following relations:

1. For the first query of $(T, M_L \| M_R)$ yielding $(C_L \| C_R)$,
$$E_{K_e}(M_L) \oplus F_{K_h}(T, M_R) = E_{K_e}(C_L) \oplus F_{K_h}(T, C_R)\,.$$
2. For the second query of $(T, M_L \| C_R)$ yielding $(M'_L \| M'_R)$,
$$E_{K_e}(M_L) \oplus F_{K_h}(T, C_R) = E_{K_e}(M'_L) \oplus F_{K_h}(T, M'_R)\,.$$
3. For the third query of $(T, C_L \| M_R)$ yielding $(C'_L \| C'_R)$,
$$E_{K_e}(C_L) \oplus F_{K_h}(T, M_R) = E_{K_e}(C'_L) \oplus F_{K_h}(T, C'_R)\,.$$

XORing these three equations together gives:

$$E_{K_e}(M'_L) \oplus F_{K_h}(T, M'_R) = E_{K_e}(C'_L) \oplus F_{K_h}(T, C'_R)\,.$$

Finally, applying the function G to both sides, we get:

$$G_{K_c,m-n}(E_{K_e}(M'_L) \oplus F_{K_h}(T, M'_R)) = G_{K_c,m-n}(E_{K_e}(C'_L) \oplus F_{K_h}(T, C'_R))$$
$$C_R \oplus M'_R = M_R \oplus C'_R\,.$$

This simplifies to $M_R \oplus C_R = M'_R \oplus C'_R$ which always holds in HCI with probability 1. For distinct queries (i.e., when $M_R \neq C_R$ and $M_L \neq C_L$), this relation is nontrivial, as in a true random involution it occurs with only a small probability (such as $\leq 2^{-n+1}$ when $m \geq 2n$) for any relevant values of n. Consequently, this allows an adversary to distinguish HCI from a random involution with probability close to 1.

Remark. This attack does not apply to GEM [17] and to any XCB variant that is not an involution as it requires both hash functions and both block ciphers to be identical.

7 Conclusion and Open Problems

This study exposes fundamental weaknesses in the XCB design strategy that were left unnoticed and reproduced in XCB-based designs for more than 20 years.

Our *shared difference attack* breaks XCB-AES in just *two queries*, invalidating its claimed sprp, stprp, and vil-stprp security.

We show how our attack easily applies to other XCB variants – XCBv1, HCI, and MXCB, invalidating *all* existing security proofs and raising practical concerns for real-world deployments of XCB-AES and XCB-style TEMs in secure storage systems. Our *flipped parts attack* breaks HCI even with inseparable hash functions.

Our attacks point to countermeasures that have been already implicitly deployed in subsequent Hash-CTR-Hash TEMs like HCTR2.

Several open problems remain:

1. Secure Involution TEM Design: Our results break HCI, which was one of the very few TEMs with the involution property. Can an involution-based TEM be designed while preserving vil-tprs security guarantees?
2. Other Potential Exploits: Our attack specifically targets XCB-style modes with polynomial hashing. Are there other classes of TEMs or symmetric designs vulnerable to similar structural weaknesses?
3. Beyond Birthday Bound (BBB) Secure Hash-CTR-Hash TEMs: HCTR2 offers a provably secure alternative to XCB, but its security is limited to the birthday bound. Can we design more robust TEMs that retain the efficient Hash-CTR-Hash structure while achieving BBB security? The approach taken in the GEM work [17] represents an ongoing effort in this direction.

Our findings stress the need for revisiting the standardization of XCB-AES[7] and similar TEMs. As disk encryption and secure storage applications increasingly rely on tweakable enciphering modes, ensuring their robustness against practical attacks remains a critical research direction.

Acknowledgements

Amit Singh Bhati was supported by CyberSecurity Research Flanders with reference number VR20192203, in part by the Research Council KU Leuven C1 on Security and Privacy for Cyber-Physical Systems and the Internet of Things with contract number C16/15/058 and by the Flemish Government through FWO Project G.0835.16 A security Architecture for IoT. Elena Andreeva was supported in part by the Austrian Science Fund (FWF) SFB project SPyCoDe 10.55776/F85.

References

1. Andreeva, E., Bhati, A.S., Preneel, B., Vizár, D.: 1, 2, 3, Fork: Counter Mode Variants based on a Generalized Forkcipher. IACR Trans. Symmetric Cryptol. **2021**(3), 1–35 (2021) 22
2. Andreeva, E., Bhati, A.S., Vizár, D.: Nonce-Misuse Security of the SAEF Authenticated Encryption mode. In: Selected Areas in Cryptography. pp. 512–534. Springer (2020) 22
3. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: How to Securely Release Unverified Plaintext in Authenticated Encryption. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 105–125. Springer (2014) 28
4. Andreeva, E., Cogliati, B., Lallemand, V., Minier, M., Purnal, A., Roy, A.: Masked Iterate-Fork-Iterate: A new Design Paradigm for Tweakable Expanding Pseudorandom Function. In: 22nd International Conference on Applied Cryptography and Network Security (2024) 22
5. Andreeva, E., Deprez, A., Pittevils, J., Roy, A., Bhati, A.S., Vizár, D.: New results and insighs on forkae. In: NIST LWC workshop (2020) 22
6. Andreeva, E., Lallemand, V., Purnal, A., Reyhanitabar, R., Roy, A., Vizár, D.: Forkcipher: a New Primitive for Authenticated Encryption of Very Short Messages. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 153–182. Springer (2019) 22
7. Andreeva, E., Weninger, A.: A forkcipher-based pseudo-random number generator. In: International Conference on Applied Cryptography and Network Security. pp. 3–31. Springer (2023) 22
8. Ashur, T., Dunkelman, O., Luykx, A.: Boosting authenticated encryption robustness with minimal modifications. In: Annual International Cryptology Conference. pp. 3–33. Springer (2017) 28

[7] IEEE Security in Storage Working Group (SISWG) has initiated a revision of the 1619.2 standard to remove XCB-AES and has submitted a project authorization request to the IEEE New Standards Committee (NesCom) for approval. [33]

9. Bellare, M., Rogaway, P.: Encode-Then-Encipher Encryption: How to Exploit Nonces or Redundancy in Plaintexts for Efficient Cryptography. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 317–330. Springer (2000) 2, 28
10. Bellare, M., Rogaway, P.: Code-based game-playing proofs and the security of triple encryption. EUROCRYPT 2006 (2006) 28
11. Bernstein, D.J.: Cryptographic competitions: CAESAR. http://competitions.cr.yp.to 2, 28
12. Berti, F., Standaert, F.X., Levi, I.: Authenticity in the presence of leakage using a forkcipher. Cryptology ePrint Archive, Paper 2024/1325 (2024) 22
13. Bhati, A.S., Andreeva, E., Müller, S., Vizar, D.: Sonikku: Gotta Speed, Keed! A Family of Fast and Secure MACs. Cryptology ePrint Archive, Paper 2024/1980 (2024) 22
14. Bhati, A.S., Andreeva, E., Vizár, D.: OAE-RUP: a strong online AEAD security notion and its application to SAEF. In: International Conference on Security and Cryptography for Networks. pp. 117–139. Springer (2024) 22
15. Bhati, A.S., Dufka, A., Andreeva, E., Roy, A., Preneel, B.: Skye: An Expanding PRF based Fast KDF and its Applications. In: 19th ACM ASIA Conference on Computer and Communications Security. ACM (2024) 22
16. Bhati, A.S., Pohle, E., Abidin, A., Andreeva, E., Preneel, B.: Let's Go Eevee! A Friendly and Suitable Family of AEAD Modes for IoT-to-Cloud Secure Computation. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. pp. 2546–2560 (2023) 22
17. Bhati, A.S., Verbauwhede, M., Andreeva, E.: Breaking, Repairing and Enhancing XCBv2 into the Tweakable Enciphering Mode GEM. Cryptology ePrint Archive, Paper 2024/1554 (2024) 22, 24
18. Bhaumik, R., List, E., Nandi, M.: ZCZ–Achieving n-bit SPRP Security with a Minimal Number of Tweakable-block-cipher Calls. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 336–366. Springer (2018) 2
19. Chakraborty, D., Ghosh, S., López, C.M., Sarkar, P.: FAST: disk encryption and beyond. Cryptology ePrint Archive (2017) 2
20. Chakraborty, D., Hernandez-Jimenez, V., Sarkar, P.: Another look at XCB. Cryptography and Communications **7**, 439–468 (2015) 1, 8, 10, 11, 13, 18
21. Chang, D., Datta, N., Dutta, A., Mennink, B., Nandi, M., Sanadhya, S., Sibleyras, F.: Release of unverified plaintext: Tight unified model and application to ANY-DAE. IACR Transactions on Symmetric Cryptology pp. 119–146 (2019) 28
22. Cogliati, B., Ethan, J., Lallemand, V., Lee, B., Lee, J., Minier, M.: CTET+: A beyond-birthday-bound secure tweakable enciphering scheme using a single pseudorandom permutation. IACR Transactions on Symmetric Cryptology **2021**(4), 1–35 (Dec 2021) 2
23. Crowley, P., Huckleberry, N., Biggers, E.: Length-preserving encryption with HCTR2. Cryptology ePrint Archive (2021) 2, 21
24. Datta, N., Dutta, A., List, E., Mandal, S.: FEDT: Forkcipher-based leakage-resilient beyond-birthday-secure AE. IACR Communications in Cryptology (2024) 22
25. Degabriele, J.P., Stam, M.: Untagging tor: a formal treatment of onion encryption. In: Advances in Cryptology–EUROCRYPT 2018. pp. 259–293. Springer (2018) 28
26. Gueron, S., Lindell, Y.: GCM-SIV: full nonce misuse-resistant authenticated encryption at under one cycle per byte. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 109–119 (2015) 12

27. Halevi, S.: EME*: Extending EME to handle arbitrary-length messages with associated data. In: International Conference on Cryptology in India. pp. 315–327. Springer (2004) 2, 3
28. Halevi, S.: Invertible universal hashing and the TET encryption mode. In: Advances in Cryptology-CRYPTO 2007. pp. 412–429. Springer (2007) 2
29. Halevi, S., Rogaway, P.: A parallelizable enciphering mode. In: Cryptographers' Track at the RSA Conference. pp. 292–304. Springer (2004) 2
30. Hoang, V.T., Krovetz, T., Rogaway, P.: Robust Authenticated-Encryption AEZ and the Problem That It Solves. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 15–44. Springer (2015) 2, 28
31. Hoang, V.T., Krovetz, T., Rogaway, P.: AEZ v5: Authenticated Encryption by Enciphering. CAESAR Competition (2017) 2
32. IEEE 1619.2: IEEE standard for wide-block encryption for shared storage media (2011) 3, 8
33. IEEE Security in Storage Working Group (SISWG) (2025), https://sagroups.ieee.org/siswg/ 25
34. Khairallah, M.: CLRW1[3] is not Secure Beyond the Birthday Bound: Breaking TNT with $O(2^{n/2})$ queries. Cryptology ePrint Archive (2023) 10
35. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. In: Advances in Cryptology-Crypto 2002, Proceedings. vol. 2442, pp. 31–46 (2002) 10
36. McGrew, D., Viega, J.: The Galois/counter mode of operation (GCM). submission to NIST Modes of Operation Process (2004) 12
37. McGrew, D.A., Fluhrer, S.R.: The extended codebook (XCB) mode of operation. Cryptology ePrint Archive (2004) 1, 2, 8, 16
38. McGrew, D.A., Fluhrer, S.R.: The security of the extended codebook (XCB) mode of operation. In: Selected Areas in Cryptography. pp. 311–327. Springer (2007) 2, 3, 8, 11, 13
39. Mouha, N., Dworkin, M.: Report on the Block Cipher Modes of Operation in the NIST SP 800-38 Series. Tech. rep., National Institute of Standards and Technology (2023) 2
40. Nandi, M.: An Efficient SPRP-secure Construction based on Pseudo Random Involution. Cryptology ePrint Archive (2008) 2, 3, 18, 21
41. Naor, M., Reingold, O.: Constructing pseudo-random permutations with a prescribed structure. Journal of Cryptology **15**, 97–102 (2002) 17
42. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 373–390. Springer (2006) 28
43. Sarkar, P.: Improving upon the TET mode of operation. In: Information Security and Cryptology-ICISC 2007: 10th International Conference, Seoul, Korea, November 29-30, 2007. Proceedings 10. pp. 180–192. Springer (2007) 2
44. Shrimpton, T., Terashima, R.S.: A modular framework for building variable-input-length tweakable ciphers. In: Advances in Cryptology-ASIACRYPT 2013: 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I 19. pp. 405–423. Springer (2013) 21
45. The23rd Raccoon: Analysis of the relative severity of tagging attacks (March 2012), http://archives.seul.org/or/dev/Mar-2012/msg00019.html 28
46. Wang, P., Mao, S., Xu, R., Jing, J., Wang, Y.: How to Recover the Full Plaintext of XCB. Cryptology ePrint Archive, Paper 2024/1527 (2024) 4, 21

A Applications of vil-stprp Enciphering Schemes

vil-stprp is the strongest security notion for TEMs. vil-stprp modes are particularly suitable for applications like packet-based communication in internet traffic, where encrypting variable-length data without ciphertext expansion is essential.

vil-stprp versus IND-CCA Security. Compared to the standard IND-CCA security notion for randomized encryption, vil-stprp (which, for a minimum input length m, is indistinguishable from an $m/2$-bit secure variable-input-length pseudorandom function (VIL-PRF) of the same size due to the standard PRP-PRF switching lemma [10]) is weaker only in the sense that, for a given key K, the same tweak-input pair (T, M) always produces the same output C. That is, a vil-stprp scheme is a deterministic algorithm and thus inherently leaks input repetitions.

However, if the tweak input is unique for repeated messages M, any vil-stprp scheme also achieves IND-CCA security. In packet-based communication protocols, the tweak can be derived from auxiliary public information such as associated data (AD), packet headers, or packet numbers, combined with a counter or nonce to ensure uniqueness.

vil-stprp and RAE Security. Robust authenticated encryption with decryptional leakage simulator ($\mathrm{RAE}_{\mathsf{sim}}$ or RAE) [30] is a strong AEAD security notion designed to meet defense-in-depth [11] requirements, such as those in Tor [25], where recent crypto-tagging attacks [45] have demonstrated vulnerabilities arising from decryption leakages that compromise anonymity. Prior research [8,21,30] has established that an RAE-secure scheme is resistant to nonce misuse (MRAE [42]) and remains secure against decryptional leakage of unverified plaintexts (INT-RUP+PA2) [3].

It was shown in [30] that an Encode-then-Encipher (EtE) [9] scheme based on a vil-stprp is RAE-secure while incurring only the minimal ciphertext expansion required for the authentication tag. This underscores the significance of vil-stprp schemes.

Encode-then-Encipher (EtE) [9], introduced by Bellare et al. in 2000, is an authenticated encryption paradigm based on an injective encoding function $\mathsf{encode}_\lambda : \{0,1\}^m \to \{0,1\}^{m+\lambda}$ (e.g., $\mathsf{encode}_\lambda(M) = 0^\lambda \| M$) and a TEM $\Pi : \mathcal{K} \times \mathcal{T} \times \{0,1\}^{m+\lambda} \to \{0,1\}^{m+\lambda}$. Given a message $M \in \{0,1\}^m$, key $K \in \mathcal{K}$, and tweak $T \in \mathcal{T}$ (which includes public inputs such as associated data A, nonce N, or initialization vector IV), encryption is defined as:

$$C = \Pi(K, T, \mathsf{encode}_\lambda(M))$$

where λ is the ciphertext expansion i.e., $|C| - |M|$.

Wide (Tweakable) Block Ciphers. TEMs with a fixed but large input length (e.g., 256 bits or more) are known as wide (tweakable) block ciphers. These ciphers, when shown secure as stprps, are deployed in file encryption and sector-oriented storage media encryption (also known as the disk-sector encryption).

They provide a strong layer of security against unauthorized access and data breaches.

Due to their efficiency in handling large data blocks, wide tweakable block ciphers are particularly well-suited for full-disk encryption. They enable fast and secure encryption and decryption operations, ensuring minimal impact on system performance while maintaining strong security guarantees.