Blockcipher-Based Key Commitment for Nonce-Derived Schemes

Panos Kampanakis[1], Shai Halevi[1][0000-0003-3432-7899], Nevine Ebeid[1], and Matt Campagna[1]

Amazon Web Services
{kpanos,shaihal,nebeid,campagna}@amazon.com

Abstract. AES-GCM has seen great adoption for the last 20 years to protect data in various use-cases because of its optimal performance. It has also posed some challenges to modern applications due to its nonce, block size, and lack of key commitment. Nonce-derived schemes address these challenges by deriving a different key from random values and using GCM with the derived key. In this work, we explore efficient key commitment methods for nonce-derived schemes. For concreteness, we focus on expanding XAES-256-GCM, a derived key scheme originally introduced by Filippo Valsorda. We propose an efficient CMAC-based key commitment solution, and prove its security in the ideal-cipher model. This proposal yields a FIPS-compliant mode and offers much better data bounds than GCM. Finally, we benchmark the new mode's performance to demonstrate that the additional cost affects mostly small plaintexts.

Keywords: XAES · KC-XAES · Key Committing AEAD · CMAC based key commitment.

1 Introduction

AES-GCM has been the status quo for efficient symmetric encryption for decades. As technology and cryptographic applications evolved over time, AES-GCM has posed some challenges to certain use-cases due to its default 96-bit nonce size, 128-bit block size, and lack of key commitment. Some of these challenges, in the context of demanding applications in cloud environments, were discussed in [21].

Nonce-derived schemes are one way of addressing these challenges: Such schemes derive multiple keys from nonce values, then apply standard AES-GCM with the derived keys (and possibly another 96-bit nonce). The approach overcomes the nonce length and data limit issues since each derived key is only used to encrypt a few messages. One example of a derived mode is used in the AWS Encryption SDK (ESDK) and AWS Key Management Service (KMS) [30] which employ HMAC-based PRFs for key derivation. Other examples include DNDK [16,17] and XAES-256-GCM [14]. Below we refer to the latter as XAES, for brevity. By itself, the use of nonce-derived keys does not address key commitment, however. Some schemes such as DNDK and the AWS ESDK chose to include a built-in key commitment mechanism, while others such as XAES left it out of scope.

In this work, we explore efficient key commitment methods that can be added to any nonce-derived scheme in a black-box manner. Our focus is on options that use the underlying block cipher, are efficient, and only use standard primitives which are FIPS-approved. We propose an efficient CMAC-based key commitment solution, and prove its security in the ideal-cipher model. We argue that adding this solution to XAES yields a FIPS-compliant mode, quantify the data and message length limits of this mode and compare this combination to other nonce-derived schemes. We also benchmark our key committing XAES performance.

Recently, after its Accordion mode workshop [26], US National Institute of Standards and Technology (NIST) has been exploring a new wide-block cipher [28] which could be used in a new AEAD mode [27] to alleviate some of AES-GCM challenges [21]. The blockcipher-based key commitment proposed in this work is generic and could be used in any new wide-block derived key mode variant by leveraging similar constructions with a wide block cipher instead of AES256.

1.1 Related Work

As we said above, there has been significant work in the literature trying the address the practical shortcomings of GCM. Prior works that use GCM in a nonce-derived mode include AWS' implementation that addressed key rotation at scale using a solution based on HMAC-SHA256 [7], Gueron's DNDK-GCM mode [16], Bhaumik and Degabriele's strong blockcipher-based PRF for GCM derived key modes [6], and Valsorda's XAES mode [14,13]. XAES is the focus of our work. Gueron and Risternpart subsequently improved DNDK by introducing DNDK version 2 [17,18] which uses less blockcipher calls to derive the key and commitment string. One more related proposal is the new accordion mode AES-GEM [3], which has large nonces, a derived key and a tweaked 64-bit internal AES-GCM counter.

Another relevant area deals with key (and context) commitment [24,1,22,4], with some works showing that GCM does not provide these properties and proposing various solutions. For example, [15] introduced key commitment to AES-GCM by using a pseudorandom function to generate a derived key and a key commitment string from the main key and a random value. The AWS ESDK and AWS KMS added key commitment in 2020 based on this work [30]. DNDK [16] and DNDK version 2 [17,18] introduced key commitment by using a beyond birthday bound PRF and a large nonce.

1.2 Organization

The rest of this document is organized as follows: In Section 2 we recall the notions of key commitment security, in Section 3 we review Valsorda's XAES mode [14] and describe our construction for adding key commitment, and in Section 4 we discuss the usage of this mode, its data limits, and FIPS-comliance.

We prove the security of our key commitment function in Section 5, and provide performance numbers in Section 6. The Appendices provide more details about the implementation of the schemes and other key commitment options we considered.

2 Key Commitment

Standard notions of security for authenticated encryption (AEAD, e.g., [5]) assume that all keys are chosen at random by legitimate users, but there are settings in which the entity choosing the keys may itself be adversarial. Perhaps surprisingly, [12] showed that AES-GCM and other common authenticated encryption schemes are susceptible to such attacks: It is rather easy to come up with two different keys and a ciphertext which would pass the authentication test relative to both keys and decrypt to two different plaintexts. It turns out that some meaningful notions of security can be achieved even in these cases. Specifically, Farshim et al. [12] and follow up works [15,22,4,1,29] described a *key commitment* property, where even a key-choosing adversary cannot come up with a ciphertext that decrypts to different plaintext messages under different keys.

As described in [12,1], key commitment turns out to be important in several real-world settings where an adversary may be able to trick different parties into using different keys to decrypt the same ciphertext. For example, in storage environments that support key rotation, an adversary could theoretically encrypt a file with one key, then rotate the key and trick the system into decrypting with the new key, causing readers to get the wrong file upon decryption. Similarly, in systems that use key-wrapping for hierarchical key management, the adversary can wrap different data keys under different high-level keys, causing different users to decrypt the same ciphertext to different plaintexts.

The literature contains a few different formal definitions for the key-commitment requirement. All of these definitions consider AEAD schemes with the following encryption/decryption interfaces:

$$\mathsf{Encrypt}(K, N, A, P) \mapsto C, \ \ \mathsf{Decrypt}(K, A, C) \mapsto P/\perp,$$

where K is the key, N is a nonce, A is some context information or additional data that needs authentication (A is called "additional authenticated data" (AAD) in AES-GCM), P is the message to be encrypted, and C is the ciphertext. These formal key commitment definitions are summarized below in order from weakest to strongest:

- The weakest notion was put forward by Farshim et al. [12]. It asserts that it is hard for the adversary to come up with two different keys, $K \neq K'$, one nonce value N, and two pairs $(A, P), (A', P')$ such that $\mathsf{Encrypt}(K, N, A, P) = \mathsf{Encrypt}(K', N, A', P')$. Below we refer to this notion as FOR-KC.
- A stronger notion called CMT-1, put forward by Bellare and Hoang [4], asserts that it is hard for the adversary to come up with two different keys, $K \neq$

K' and two tuples $(N, A, P), (N', A', P')$ such that $\mathsf{Encrypt}(K, N, A, P) = \mathsf{Encrypt}(K', N', A', P')$.
- Yet a stronger notion is CMT-2, implicit in [4] and mentioned explicitly by Takeuchi et al. [29], asserting that it is hard for the adversary to come up with two different pairs $(K, N) \neq (K', N')$ and two other pairs $(A, P), (A', P')$, such that $\mathsf{Encrypt}(K, N, A, P) = \mathsf{Encrypt}(K', N', A', P')$.
- Bellare and Hoang [4] also defined the strongest notion of this type, called CMT-4 (and also referred to as *context commitment* [24]), asserting that it is hard for the adversary to come up with two different tuples $(K, N, A, P) \neq (K', N', A', P')$ such that $\mathsf{Encrypt}(K, N, A, P) = \mathsf{Encrypt}(K', N', A', P')$. This last notion asks for all, key, nonce, additional data, and plaintext values to be committed to which differs from FOR-KC, CMT-1, and CMT-2. Achieving efficient context commitment (while keeping the ciphertext length independent of the length of A) requires two passes on the additional data and plaintext and newer constructions like truncated Davies-Meyer or iterative truncated-permutation [4], and we do not consider it in this manuscript.

FOR-KC vs. CMT-1 vs. CMT-2: While the three notions, FOR-KC, CMT-1, CMT-2, are different, the distinction turns out to be meaningless for most common AEAD use-cases. Specifically, these notions are the same if the nonce value N is part of the output ciphertext C, which is usually the case. Note, however, that the distinction could be important in applications where the nonce value is implied and is not included explicitly in the ciphertext. For example, a storage system that does not include the nonce in the ciphertext but instead uses the object index as the nonce (bad idea) could be vulnerable to key commitment attacks even if it satisfied the FOR-KC property. In this manuscript, we focus on AEAD schemes that include the nonce explicitly with the ciphertext, hence we only consider FOR-KC.

3 Adding Key Commitment to XAES

3.1 XAES

XAES-256-GCM by Filippo Valsorda [14] specified XAES as a variant of AES-GCM which takes a 192-bit (24-byte) nonce and a 256-bit main key. It derives a new AES-GCM key from the main key and half the nonce for every invocation. The other half of the nonce is used as the Initialization Vector (IV) to encrypt the message with AES-GCM and the derived key.

In more detail, the input to XAES is a 256-bit main key K, a 24-byte nonce N, Plaintext P, and Additional Authenticated Data A. The output is the ciphertext C. Denote the first 12 bytes of the nonce by U and the last 12-bytes by V. XAES uses CMAC-AES-256 to derive another 256-bit key K_U from K and U, and then encrypts the message using AES-256-GCM with key K_U and nonce V. We provide a pseudo-code below for self-containment. The two 32-bit constants in that code are $\text{const}_1 = \texttt{0x00015800}$ and $\text{const}_2 = \texttt{0x00025800}$. (Note that the GCM ciphertext includes in particular the nonce value V, so the output of XAES includes the entire nonce N.)

```
XAES-Encrypt(Key $K$, Nonce $N$, AAD $A$, Plaintext $P$):
0.  $U = N[0:12],\ V = N[12:24]$              // $U, V$ are the two halves of $N$
1.  $K1 = X \times \mathrm{AES}_K(0^{128})$            // Multiplication by $X \in GF(2^{128})$
2a. $M_1 = (\mathrm{const}_1 \parallel U) \oplus K1$
2b. $M_2 = (\mathrm{const}_2 \parallel U) \oplus K1$
3.  $K_U = \mathrm{AES}_K(M_1) \parallel \mathrm{AES}_K(M_2)$    // CMAC-based key derivation
4.  Output $U \mid \mathrm{AES\text{-}GCM}_{K_U}(V, A, P)$   // AES-GCM encryption
```

Note that the blockcipher-based key derivation in XAES is used beyond the birthday bound. In a separate analysis manuscript [2], we prove that this use of CMAC maintains the security of XAES, showing that it is safe to derive close to 2^n keys using an n-bit blockcipher in counter mode as in XAES.

3.2 KC-XAES

XAES is a derived key mode of AES-GCM. Like with plain AES-GCM, an adversary could find two different derived keys, K_U and K'_U which could have been produced from the same main key K and lead to the same ciphertext and authentication tag. Decrypting the ciphertext with the two derived keys would lead to two different plaintexts. Thus, XAES does not provide key commitment for any of the notions defined in Section 2.

One solution for derived key modes is PRF-based key commitment, as proposed by Gueron in [15], which provides key-commitment by using a hash-based PRF (e.g., HMAC-SHA256), at the cost of the PRF performance. An alternative is to implement the generic solution of Albertini et al. [1, Sec 5.4], adding to the ciphertext a key commitment value $F_{com}(K, N)$ that is verified upon decryption. While Albertini et al. say that $F_{com}(K, \cdot)$ should be a PRF (which is independent of the one used to derive data key), it is clear that this PRF must also be collision-resistant. Specifically, for this construction to yield FOR-KC, it must be hard for an attacker to find two keys $K \neq K'$ and a nonce N such that $F_{com}(K, N) = F_{com}(K', N)$.

Our aim in this work is to describe an efficient construction using only the underlying blockcipher, so in particular we need to describe a blockcipher-based construction that offers collision-resistance and does not leak information about the main key K. To achieve this, we propose:

$$F_{com}(K, N) = CMAC_K(c\|U\|V\|c') \parallel CMAC_K(c\|U\|V\|c''),$$

where c, c', and c'' are different 32-bit constants (that are also different from the constants used for key derivation), which are defined later. With the 192-bit nonce of XAES, the length of each of the CMAC inputs $c\|N\|c'$ and $c\|N\|c''$ is 256 bits, or two AES blocks. We note that computing $F_{com}(K, N)$ only takes three more block encryption operations on top XAES, since the first block is the same in both CMAC calls. A depiction of the function $F_{com}(K, N)$ in given in Figure 1, and a pseudocode for the combined XAES with key commitment is given in Figure 2. Note that all the derived values (both derived keys and key-commitment values) are obtained by CMAC with the same main key on different inputs, so they are pseudorandom and (pseudo)independent of each other. In other words, revealing the key-commitment values does not weaken the derived keys of XAES.

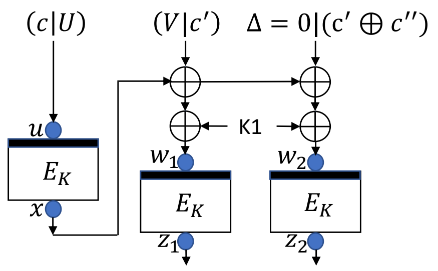

Fig. 1. The function F_{com}, applied to the 192-bit nonce $N = (U \parallel V)$. c, c', c'' are 32-bit constants. The 256-bit output is $K_C = (z_1 \parallel z_2)$.

```
KC-XAES-Encrypt(Key $K$, Nonce $N$, AAD $A$, Plaintext $P$):
0.  $U = N[0:12]$, $V = N[12:24]$                      // $U, V$ are the two halves of $N$
1.  $K1 = X \times \mathrm{AES256}_K(0^{128})$                 // Multiplication by $X \in GF(2^{128})$
2a. $M_1 = (\mathrm{const}_1 \parallel U) \oplus K1$                  // CMAC-based key derivation
2b. $M_2 = (\mathrm{const}_2 \parallel U) \oplus K1$
3.  $K_U = \mathrm{AES256}_K(M_1) \parallel \mathrm{AES}_K(M_2)$          //$K_U = CMAC_K(\mathrm{const}_1 \| U) \parallel CMAC_K(\mathrm{const}_2 \| U)$
4.  $CT = \mathrm{AES256\text{-}GCM}_{K_U}(V, A, P)$               // AES-GCM encryption
5.  $X_1 = \mathrm{AES256}_K(\mathrm{const}_3 \parallel U)$                // CMAC-based key commitment
6a. $W_1 = X_1 \oplus (V \parallel \mathrm{const}_4) \oplus K1$
6b. $W_2 = X_1 \oplus (V \parallel \mathrm{const}_5) \oplus K1$
7.  $K_C = \mathrm{AES256}_K(W_1) \parallel \mathrm{AES256}_K(W_2)$    // $K_C = CMAC_K(\mathrm{const}_3 \| U \| V \| \mathrm{const}_4)$
8.  Output $U \| CT \| K_C$                              //        $\parallel CMAC_K(\mathrm{const}_3 \| U \| V \| \mathrm{const}_5)$
```

Fig. 2. The KC-XAES256 Specification. The constants are defined as const_1 = 0x00015800, const_2 = 0x00025800, $\mathrm{const}_3 = c =$"XCMT"= 0x58434D54, $\mathrm{const}_4 = c' =$ 0x00010001, $\mathrm{const}_5 = c'' =$ 0x00010002. Prepending constant const_3 to the input of $CMAC_K$ and appending const_4 and const_5 allows us to re-use X_1 and omit an extra AES256 call while remaining FIPS compliant as per [9].

Appendix A breaks down Figure 2 into a prescriptive implementation specification. Appendix B abstracts the specification to allow for smaller nonces useful to constrained use-cases and Appendix C lays out other $F_{com}(K, N)$ options we considered and did not pick.

4 KC-AES Usage and Data Limits

We analyze below the usage, data limits, and FIPS certification considerations for the KC-XAES construction as specified in Figure 2.

4.1 Uses

XAES-256-GCM [14] introduces three additional AES-256 calls on top of AES-256-GCM (one of which can be computed only once and cached for all the XAES calls with the same main key). Our CMAC-based key commitment adds three more AES-256 calls. The impact of these extra calls is very small when we use this mode to encrypt large messages, since we anyways need many AES-256 calls to produce the ciphertext.

Some of the use-cases that could benefit from XAES are described in [21]. These include high encryption rate uses that can run through 2^{32} 96-bit AES-GCM IVs in seconds and where deterministic IVs pose challenges due to their statefulness. Other uses include encryption clients and servers like the AWS ESDK with AWS Key Management Service that encrypt many messages from one main key and require key commitment without maintaining state.

4.2 Data Limits

XAES is a nonce-derived mode that derives a random data-key K_U and IV for GCM per invocation. Hence XAES remains as secure as AES-GCM as long as the pair (K_U, IV) does not repeat, and as long as the data bounds per key K_U do not violate AES-GCM's limits. There are various works that analyze the security of AES-GCM which set the maximum data bounds based on the desired confidentiality and integrity level [19,20,23]. These bounds still apply to XAES and we consider them below when we calculate the maximum data size per derived key K_U. Additionally, a separate manuscript [2] proves that using blockcipher-based key derivation beyond the birthday bound maintains the security of the scheme to 2^n keys for n-bit blockcipher in counter mode.

First, we focus on the (K_U, V) collision probability. Clearly, the pair (K_U, V) would repeat if the 24-byte nonce N was re-used with the same main key K. As long as the nonces N do not collide, we will not have (K_U, V) collisions under the same main key K. To see that, observe that since AES is a permutation, then the two blocks of K_U can only collide if the corresponding AES inputs (M_1, M_2 from Figure 2) repeat. $K1$ is fixed for the same K which means we can have a K_U collision (under the same main key K) only when we have a U collision. Thus, a (K_U, V) collision requires a full nonce (U, V) collision.[1]

It follows that when the nonce N is chosen at random, a full (K_U, V) collision will only happen after roughly $2^{|N|/2} = 2^{96}$ calls. If we want to limit the probability of such collisions to less than 2^{-32}, then it is sufficient to limit the number

[1] While it is theoretically possible to have a (K_U, V) collision under *different* main keys $K \neq K'$, if these main keys are chosen at random then the K_U's will be pseudorandom and so will not collide except with insignificant probability.

of calls below 2^{80}. To see that, note that by the union bound the probability of having a collision among a set of 2^{80} random 192-bit (24-byte) strings is at most

$$\binom{2^{80}}{2} \cdot 2^{-192} \leq 2^{2\cdot 80-1} \cdot 2^{-192} = 2^{-33}.$$

An AES-GCM (key, IV) collision probability $\leq 2^{-32}$ is typically considered satisfactory. It was initially introduced by NIST in [11] and it usually dictates AES-GCM's rekey frequency when IVs are generated at random. We note that while the analysis above rules out full-nonce collisions, we still could have collisions for $U = N[0:12]$, the portion of the nonce that is used for key-derivation. Collisions like that would have us use the same data key K_U to encrypt multiple messages, which is not a problem as long as these different encryptions use different GCM nonces V (which is ensured by the analysis above). By the analysis above, we can use each main key K to encrypt at most 2^{80} messages.

Let m be a parameter (to be determined later), then the probability of the same value $U \in \{0,1\}^{96}$ occurring m times among 2^{80} messages (assuming random nonces) is bounded using the union bound by

$$\binom{2^{80}}{m} \cdot 2^{-96(m-1)} \approx \frac{2^{80m-96(m-1)}}{m!} = \frac{2^{96-16m}}{m!}. \tag{1}$$

Setting $m = 8$, eq. (1) tells us that the probability of seeing an eight-way collision is bounded by $2^{96-(96-80)\cdot 8}/8! \approx 2^{-47.3}$. Hence we can safely assume that we will not see more than seven collisions (with probability $> 2^{-32}$), which means that no derived data-key K_U will be used to encrypt more than seven messages. In settings where we can enforce fewer messages per main key, the K_U occurrence bound m is lower.

Some applications need to enforce usage limit for keys (say, no more than X-bytes-per-GCM-key). In such applications, we need to ensure that no single data key K_U is used to encrypt more than X bytes. The bound from eq. (1) above implies that except with probability $\epsilon < 2^{-47}$, we will only see at most $m-1 = 7$ messages encrypted under the same data key K_U. Hence, an application that wants to enforce X-bytes-per-GCM-key limit can achieve this by limiting the length of the encrypted messages to no more than $X/7$ bytes each. (Note there is still an inherent limit on each AES-GCM invocation where the plaintext cannot exceed $2^{36} - 32 \approx 2^{36}$ bytes [11, § 5.2.1.1]. Thus, the KC-XAES message limit ought to be bound by $\min(X/7, 2^{36})$ bytes). As an example, NIST [11] defines a data limit of 2^{64} blocks per GCM key which means $X = 2^{68}$. In that case, each message could be up to 2^{36} bytes ($\ll 2^{68}/7$). Other X values for AES-GCM depending on the attack success probability can be found in [23] which calculates the bounds based on the analysis in [20]. [19] includes a tighter AES-GCM data bound analysis.

Are 2^{80} messages enough? Some prior work such as [21] expressed a desire to support as many as 2^{112} messages encrypted under a single key (which is what we would get from a straightforward adaptation of GCM mode to a 256-bit block cipher). XAES with 24-byte random nonces does not satisfy this desire because, as we point out above, it only supports up to 2^{80} messages under a single main key. Nonetheless, 2^{80} messages is still much better than what we have today with AES-GCM, and it seems sufficient for most of today's use-cases. Indeed, for most of today's workloads it is enough to encrypt only 2^{64} messages under one key.

Multi-key XAES: The analysis above is only concerned with the case where a single XAES main key is used for encrypting many messages. When dealing with multiple keys, ensuring a small enough collision probability requires that we bound not only the number of messages per key but also the total number of messages under all keys. For example, if we use XAES 24-byte random nonces and limit ourselves to 2^{64} messages per main key, then for each key we get a nonce N collision probability of at most 2^{-65}, and we can use as many as 2^{33} keys for a total of $2^{64+33} = 2^{97}$ messages before the multi-key-IV collision probability increases to 2^{-32}. Note that GCM's multi-key-IV collision probability is far worse. It becomes 2^{-32} after using just two keys to encrypt 2^{32} messages with each key (assuming random 96-bit IVs).

4.3 FIPS Compliance

Here we argue that KC-XAES falls under the NIST FIPS 140-3 program [25] approved algorithms, and systems that use it could be FIPS validated. FIPS 140-3 defines a set of approved cryptographic algorithms and certifies their implementations. Only certified cryptographic modules and approved algorithms can be used in Federal Government systems in the USA and Canada. Other government and International requirements like Common Criteria [10] sometimes require FIPS too.

Valsorda explains briefly in [14] how XAES maps to FIPS-compliant key derivation and use of AES-GCM. We argue that KC-XAES can be used in environments that require FIPS compliance by showing that the derived K_U key, the IV used in the AES-GCM calls of XAES, and the derived K_C follow FIPS requirements.

In KC-XAES, a new 256-bit data key K_U is derived from the main key K as

$$\begin{aligned} K_U = {} & \text{CMAC-AES256}_K(0\text{x}00015800 \| N[:12]) \ \| \\ & \text{CMAC-AES256}_K(0\text{x}00025800 \| N[:12]). \end{aligned}$$

This translates to a Counter Mode Key Derivation Function (KDF) as defined in NIST SP 800-108 [9], where CMAC-AES256 is the PRF, the fixed input data is $0\text{x}5800 \| N[:12]$, and the counter $i = 1, 2$ is two bytes. Thus, the derived key is FIPS approved as a Key Based Key Derivation Function (KBKDF) specified in [9, §4.1]. Note that one issue with key derivation using CMAC-AES256 discussed in [9, §6.7] is Key Control Security for inputs that span more than one 128-bit blocks. XAES' derived key K_U does not have this concern because the CMAC input is always one block.

Subsequently, KC-XAES uses the derived key K_U and the last 12-bytes of the nonce $V = N[12 : 24]$ in AES-GCM-256. If $N[12 : 24]$ was generated by using a FIPS-approved entropy source (or any other approved way of generating nonces), then the KC-XAES encryption is FIPS approved because it encrypts with AES-GCM which is FIPS-approved and a key K_U and IV generated with FIPS-approved methods. Of course, the message and data limits laid out in Section 4.2 apply to ensure the NIST security bounds for AES-GCM hold.

Regarding key commitment, the key commitment string generation K_C is

$$\begin{aligned} K_C = {} & \text{CMAC-AES256}_K(\text{“XCMT”} \| N[:24] \parallel \text{0x00010001}) \parallel \\ & \text{CMAC-AES256}_K(\text{“XCMT”} \| N[:24] \parallel \text{0x00010002}). \end{aligned}$$

This translates to a Counter Mode KDF as defined in [9], where CMAC-AES256 is the PRF, the fixed input data is “XCMT”$\|N[:24] \parallel$ 0x000100, and the counter $i = 1, 2$ is one byte. Thus, the derived commitment string is generated with a FIPS approved KBKDF [9, §4.1], although it admittedly is not used as an encryption key.

5 Key Commitment Security

This section uses the notations from Figure 1.

5.1 CMT-2 Insecurity

In section 5.2 below we show that in the ideal-cipher model with n-bit blockcipher, our construction offers roughly n bits of security against “FOR-KC collision attacks”, where the adversary must use the same nonce for the two keys. Before that, and in order to build some intuition, we describe an attack against the stronger CMT-2 notion, where the attacker can use different nonces with the different keys. We recall again that in most use-cases, where the nonce is included as part of the ciphertext, FOR-KC implies also CMT-1 and CMT-2. The attack below does not apply in those cases.

We now show a CMT-2 attack with complexity roughly $2^{n/2}$, even though the key-commitment length is $2n$ bits. That is, it only takes about $2^{n/2}$ queries to the blockcipher to find two different keys $K \neq K'$ and two sets of nonces U, V, U', V', such that $F_{com}(K, U, V) = F_{com}(K', U', V')$ for the function F_{com} from Figure 1. The collision attack proceeds as follows (with the the blockcipher being AES and with the constants $c, c', c'' \in \{0,1\}^{32}$): Denote $\Delta = 0|(c' \oplus c'') \in \{0,1\}^{128}$.

1. Choose an arbitrary 128-bit block $z_1 \in \{0,1\}^{128}$.
2. For $i = 1, 2, 3, \ldots$, choose a random key K^i and compute:

$$w_1^i \leftarrow AES_{K^i}^{-1}(z_1), \quad w_2^i \leftarrow w_1^i \oplus \Delta, \quad z_2^i \leftarrow AES_{K^i}(w_2^i).$$

   Repeat until you find a collision, $z_2^i = z_2^j$ for different keys $K^i \neq K^j$.
   Below we denote the colliding value by $z_2 := z_2^i = z_2^j$. We also denote $K := K^i, w_1 := w_1^i, w_2 := w_2^i$ and $K' := K^j, w_1' := w_1^j, w_2' := w_2^j$.

3. Let $K1 = X \times E_K(0)$ and $K_1' = X \times E_{K'}(0)$ (multiplication by X in $GF(2^n)$).
4. Try many values for $U \in \{0,1\}^{96}$, for each value compute $x \leftarrow AES_K(c\|U)$, until $x \oplus K1 \oplus w_1 = (V\|c')$ for some $V \in \{0,1\}^{96}$.
5. Similarly try many values for $U' \in \{0,1\}^{96}$, for each one compute $x' \leftarrow AES_{K'}(c\|U')$, until $x' \oplus K_1' \oplus w_1' = (V'\|c')$ for some $V' \in \{0,1\}^{96}$.
6. Output (K, U, V) and (K', U', V').

The complexity of step 2 is roughly 2^{64}, and the complexity of steps 4, 5 is roughly $2^{|c'|}$ (which is 2^{32} in our construction). Overall the complexity of this attack is about 2^{64}. This collision attack on the key-commitment function can then be extended to a full CMT-2 attack on XAES with this function, using the known key-commitment attacks on AES-GCM [24].

5.2 FOR-KC Security

We now show that for the weaker FOR-KC notion, when the attacker must use the same nonce for both keys, we get roughly n bits of security. We note that proving the hardness of collision-finding for a blockcipher-based construction often requires resorting to idealized models such as the ideal-cipher model, and our proof is no different. Drawing conclusions from such a proof takes some caution, especially when using AES256 for which related-key attacks are known. The specific related-key attacks on AES256 do not seem to apply directly to our construction, but more cryptanalysis may be needed before accepting the numbers of our ideal-cipher analysis. This is an interesting topic for future work.

Fix an n-bit blockcipher and some n-bit block $\Delta \neq 0$, and consider the function

$$F_{com} : \mathsf{Keys} \times (\{0,1\}^n)^2 \to \{0,1\}^{2n}, \quad F_{com}(K, U, V) = CMAC_K(U\|V) \,\| \, CMAC_K(U\|V \oplus \Delta). \tag{2}$$

(This is exactly our construction, except that in this section we give the adversary a little more freedom by not insisting on the specific constants c, c', c'' from Figure 1.) Below we prove that in the ideal-cipher model, an adversary asking $q \ll 2^n$ blockcipher queries has probability $O(q)/2^n$ of finding a collision, i.e. two keys $K \neq K'$ and inputs U, V such that $F_{com}(K, U, V) = F_{com}(K', U, V)$.

One high-level intuition is that a blockcipher query made by the adversary can only induce a collision if it satisfies four equalities over n-bit blocks (i.e., the two nonce blocks and two output blocks). The adversary can force at most two of these equalities to hold (e.g. by going backward from one of the output blocks and choosing U as in the attack from above), but this leaves two n-bit equalities to chance. After making i blockcipher applications, a new application would therefore only induce collision with probability about $i/2^{2n}$. Hence, the overall probability of finding a collision after q blockcipher queries cannot be much more than $\binom{q}{2}/2^{2n}$.

Technically, what we need to prove is that even though an attacker can induce "partial collisions" as per the attack from above with complexity $2^{n/2}$, they cannot get too many of them. In particular, in Lemma 1 we show that with high probability, no key will induce partial collisions with too many other keys. Once this is established, the rest of the proof proceeds via a straightforward (if somewhat tedious) case analysis, resulting in the following lemma:

Theorem 1. *With an n-bit blockcipher in the ideal-cipher model, an adversary making at most q blockcipher queries (in either direction) has probability at most*

$$\mathsf{adv}(q,n) = \min_{\beta\in\mathbb{N}} \left\{ \frac{q\cdot\binom{q-1}{\beta}}{(2^n-2q)^\beta} + \frac{4(\beta-1)q}{2^n-2q} \right\} + \frac{3\binom{q}{2}}{(2^n-2q)^2}$$

of finding two keys $K' \neq K$ and inputs U, V such that $F(K,U,V) = F(K',U,V)$.

For example, with $n = 128$ and $q = 2^{80}$, using $\beta = 3$ yields a bound of $\mathsf{adv}(q,n) < 2^{-44} = \frac{16q}{2^n}$.

Proof (of Theorem 1). We will use the notations from Figure 1, where for each key K we have $K1 = X \times E_K(0^{128})$, and the inputs and outputs of the block cipher are denoted u, w_1, w_2 and correspondingly x, z_1, z_2. (Presumably the inputs are computed by the adversary as $u = (c\|U)$, $w_1 = x \oplus (V\|c') \oplus K1$ and $w_2 = x \oplus (V\|c'') \oplus K1$ for the nonce value $N = (U\|V)$ and constants c, c', c''.) We also denote the 128-bit block $\Delta = 0\|(c' \oplus c'')$.

To slightly simplify the case analysis below, we consider a *principled adversary* such that: (a) they never make a redundant blockcipher query that they already know the answer to; (b) with each forward-query to a block cipher $s = E_K(a)$ the adversary also makes another forward query $s' = E_K(a \oplus \Delta)$ with the same key K; and (c) with each backward-query $a = E_K^{-1}(s)$ the adversary also makes the forward query $s' = E_K(a \oplus \Delta)$ with the same key K. Below we call the two queries $((a,s)(a',s'))$ a query pair. It is a forward pair if the first query in that pair is forward, and otherwise (if the first query is backward) it is a backward pair.

Clearly, any q-query adversary can be converted into a principled adversary making at most q pairs of queries. Also, note that query pairing is a partition of the queries relative to each key into disjoint pairs. This means that if $((a,s),(a',s'))$ is a query pair for key K, then neither (a,s) nor (a',s') belongs to any other pair for that key.

Fix a principled adversary $\mathcal{A}$, making at most q query pairs to the underlying block cipher and also fix the randomness of $\mathcal{A}$. The *transcript* of the interaction between $\mathcal{A}$ and the scheme consists of entries of the form $(K, \mathsf{dir}, (a,s), (a' = a \oplus \Delta, s'))$, where $\mathsf{dir} \in \{+,-\}$ denotes whether this is a forward or backward pair.[2] The transcript is a random variable over the choice of the random permutations in the ideal cipher model. We use the following notations:

– We write $(K, \mathsf{dir}, (a,s), (a',s')) \in \Pi$ when this query pair appears in the transcript Π. We replace some of these quantities with '$\cdot$' (or omit them completely) when we mean them to remain unspecified. For a few examples:

[2] Note that specifying a' is redundant, it is there just to aid readability.

  - We write $K \in \mathit{\Pi}$ to denote that some query pair in $\mathit{\Pi}$ was made with respect to K;
  - $(K, (a, s)) \in \mathit{\Pi}$ means that $\mathit{\Pi}$ includes either query $s = E_K(a)$ or $a = E_K^{-1}(s)$;
  - $(K, \cdot, (a_1, s_1), (a_2, s_2)) \in \mathit{\Pi}$ means that this query pair is in $\mathit{\Pi}$ without specifying the direction;
  - $(K, \{z, z'\}) \in \mathit{\Pi}$ means that $(\cdot, z), (\cdot, z')$ showed up in some query pair under K, in either order. Namely, either $(K, \cdot, (\cdot, z), (\cdot, z')) \in \mathit{\Pi}$ or $(K, \cdot, (\cdot, z'), (\cdot, z)) \in \mathit{\Pi}$.

– Two query pairs, $(K, \cdot, (\cdot, z_1), (\cdot, z_2))$, $(K', \cdot, (\cdot, z_1'), (\cdot, z_2')) \in \mathit{\Pi}$, are called a *partial collision* if $K \neq K'$ but $\{z_1, z_2\} = \{z_1', z_2'\}$ as sets, irrespective of order. These are important since a collision between K and K' requires a partial collision, in addition to using the same nonces U, V under both keys. A partial collision between two query pairs is either a *forward partial collision* or a *backward partial collision*, depending on whether the latter query pair is forward or backward.[3]

– For any key $K \in \mathit{\Pi}$, let $\mathsf{col}_{\mathit{\Pi}}(K)$ denote the set of partial collisions involving K,

$$\mathsf{col}_{\mathit{\Pi}}(K) = \{(K', \{z, z'\}) \in \mathit{\Pi} : K' \neq K, (K, \{z, z'\}) \in \mathit{\Pi}\}\,. \tag{3}$$

Let maxCol be the largest number of partial collisions for any key, $\mathsf{maxCol}(\mathit{\Pi}) = \max_{K \in \mathit{\Pi}} (|\mathsf{col}(K)|)$.

– A *full collision* between two keys $K \neq K'$ involves a partial collision $(K, \cdot, (w_1, z_1), (w_2, z_2))$, $(K', \cdot, (w_1', z_1'), (w_2', z_2')) \in \mathit{\Pi}$ with $\{z_1, z_2\} = \{z_1', z_2'\}$, and in addition queries $(K, (u, x))$, $(K', (u, x')) \in \mathit{\Pi}$ (with the same u) such that either $x \oplus K1 \oplus w_1 = x' \oplus K_1' \oplus w_1'$ or $x \oplus K1 \oplus w_1 = x' \oplus K_1' \oplus w_1' \oplus \mathit{\Delta}$. (Recall that $w_2' = w_1' \oplus \mathit{\Delta}$.)

With these notations, we prove below the following three technical claims:

Lemma 1. *For a transcript of $q < 2^{n-1}$ query pairs,*

$$\Pr[\exists \text{ any forward partial collisions}] \leq \frac{2\binom{q}{2}}{(2^n - 2q)^2}.$$

Lemma 2. *For a transcript of $q < 2^{n-1}$ query pairs and any bound $\beta \in \mathbb{N}$,*

$$\Pr[\mathsf{maxCol}_{\mathit{\Pi}} \geq \beta \text{ with no forward partial collisions}] \leq \frac{q \cdot \binom{q-1}{\beta}}{(2^n - 2q)^{\beta}}.$$

Lemma 3. *For a transcript of $q < 2^{n-1}$ query pairs and any bound $\beta \in \mathbb{N}$,*

$$\Pr[\text{any full collision with } \mathsf{maxCol}_{\mathit{\Pi}} < \beta \text{ and no forward partial collisions}]$$

$$\leq \frac{4(\beta - 1)q}{2^n - 2q} + \frac{\binom{q}{2}}{(2^n - 2q)^2}.$$

[3] It is insignificant if the earlier query pair is forward or backward, only the direction of the latter pair matters.

Theorem 1 is deduced just by adding the bounds in the three lemmas above.

Proof (of Lemma 1). A forward query pair $(K^i, +, (\cdot, z_1^i), (\cdot, z_2^i)) \in \Pi$ can only induce a partial collision with some prior query $(K^j, \cdot, (\cdot, z_1^j), (\cdot, z_2^j)) \in \Pi$ if both z_1^i, z_2^i appeared in that prior query. This being a forward query, both z_1^i and z_2^i are drawn at random from a set of size at least $2^n - 2i \geq 2^n - 2q$, so the probability that it collides with the prior pair j is bounded by $2/(2^n - 2q)^2$. (The factor 2 in the numerator is because it could collide in either order: $z_1^i = z_1^j$ and $z_2^i = z_2^j$, or $z_1^i = z_2^j$ and $z_2^i = z_1^j$.) Hence, the probability of any partial collision being induced by a forward pair is bounded by $2\binom{q}{2}/(2^n - 2q)^2$ as in Lemma 1.

Proof (of Lemma 2). For any specific key K, a *neighboring query pair* to K (or just a neighbor, for short) is a query pair using some other key $K' \neq K$, which is involved in a partial collision with K. Our goal is to bound the probability of any key K having β neighbors, where all the partial collisions are induced by backward queries.

We start by numbering all the keys that are used in the transcript (e.g., by the order in which they appeared there), then fix one of these keys (call it K^*) by choosing its number $i \leq q$. We also fix a set of β query pairs *under keys other than* K^* by choosing $j_1 < j_2 < \cdots < j_\beta < q$ out of the indexes of pairs that are not using K^*. There are at most $q \cdot \binom{q-1}{\beta}$ ways to select this key and those pairs. Below we prove that for each such choice, the probability of all these β query pairs being neighbors of K^* is bounded by $1/(2^n - 2q)^\beta$. The claim then follows by the union bound.

To prove our bound, we build a labeled directed graph whose nodes are (the indexes of) all these β query pairs, as well as all the query pairs under the key K^*. We start by having a fixed set of (indexes of) query pairs $J = \{j_1, j_2, \ldots, j_\beta\}$, and a fixed (index of) another query pair i that determines the key K^*. Initially the nodes of this graph include only the set J. Every time the adversary makes a query pair under K^*, then the index of that query pair is added as another node to the graph. We label each node by the key that was used in the corresponding query pair. A directed edge $m \to \ell$ is *tentatively* added to the graph if all the following conditions hold:

- $m > \ell$;
- the m-th pair is a backward query pair, $(K^m, -, (\cdot, z_1^m), (\cdot, \cdot))$;
- the ℓ-th query pair is $(K^\ell, \cdot, (\cdot, z_1^\ell), (\cdot, z_2^\ell))$ with $K^\ell \neq K^m$ and $z_1^m \in \{z_1^\ell, z_2^\ell\}$;
- the ℓ-th query pair has no other incoming edges.

Intuitively, the adversary is making the query pair m in an attempt to create a partial collision with the query pair ℓ, starting with a backward query with one of the blocks in the ℓ-th pair. The edge becomes *permanent* if the attempt is successful, namely $\{z_1^m, z_2^m\} = \{z_1^\ell, z_2^\ell\}$, which happens with probability at most $1/(2^n - 2q)$, independently of anything that had happened before. Otherwise the edge is removed from the graph and the adversary is moving to the next query pair.

This process continues until the transcript is done, at which point the neighbors of K^* from among the nodes J are those that are connected to some K^*-labeled node in the underlying undirected graph. A crucial observation is that any tentative edge in this process that does not turn into a permanent edge, implies that the nodes in J cannot all be neighbors of K^*. Recall that a tentative edge means that the two nodes share one of the z blocks, and it fails to turn into a permanent edge if they do not share also the other z block.

Consider the case where one of these nodes (call it k^*) is labeled by K^* and the other is $j \in J$ which is labeled by another key $K \neq K^*$. Then no node labeled by K^* will be connected to node j in the underlying undirected graph, because the node k^* is the only K^*-labeled node that has the z block on which they agree, and the other z block in that node differs.

For the same reason, if two nodes that are labelled by two keys other than K^* had a tentative edge that did not turn into a permanent edge, then a K^*-labeled node can be a neighbor of one of them or the other, but then the node that was left out cannot be a neighbor of any K^*-labelled node.

Therefore, assuming no forward partial collisions, the only way in which this process ends up with all the nodes in J being neighbors of K^* is if the graph only ever had β tentative edges that all turned into permanent edges. This happens with probability at most $1/(2^n - 2q)^\beta$, which is what we needed to show for Lemma 2.

Proof (of Lemma 3). Consider a particular query pair i in Π, labeled by key K, and we bound the probability that it is the first query that induces a full collision. The full collision induced by this pair includes in particular a partial collision, either between two prior query pairs or between the i-th query pair itself and a prior pair. Either way, one of the keys in that partial collision is K and the other is some $K' \neq K$ (or else the i'th pair could not have induced a full collision). We analyze these two cases separately.

Case 1. A prior partial collision: In this case the partial collision happened before the i-th query pair. Note that before the i-th query pair, the key K could have been involved in several partial collisions, but no more than $\beta - 1$ of them. Below we fix one of them (say between the query pairs j and j') and bound the probability that the i-th query pair completes that partial collision into a full collision.

For this case we denote the i-th query by $(K = K^i, \mathsf{dir}^i, (u_1^i, x_1^i), (u_2^i, x_2^i))$ (with $u_2^i = u_1^i \oplus \Delta$), and denote the prior partial collision involving K by $(K = K^j, \cdot, (w_1^j, z_1), (w_2^j, z_2))$ and $(K' = K^{j'}, \cdot, (w_1^{j'}, z_1), (w_2^{j'}, z_2))$, with the same z_1, z_2 but $K' \neq K$. For the i-th query pair to complete this partial collision into a full collision, it must be the case that:

– The same u block is queried under K and K', so at least one of u_1^i and u_2^i must have been queried as a pre-image of $E_{K'}(\cdot)$. Since this is a principled adversary, it means that they both appears in the same query pair (since pre-images always come in pairs that are Δ apart).

Let $(K' = K^{i'}, \cdot, (u_1^{i'}, x_1^{i'}), (u_2^{i'}, x_2^{i'}))$ be query pair i' where theses values were queried with key K', to have full collision we must have $\{u_1, u_2\} = \{u_1^{i'}, u_2^{i'}\}$ (as sets, in either order).

- A full collision means in particular the same block v was used, so at least one of the following equalities must hold:

$$\begin{array}{lll} 1.\ x_1^i \oplus w_1^j \oplus K1 = x_1^{i'} \oplus w_1^{j'} \oplus K_1' & \#\ u = u_1^i,\ v \in \{x_1^i \oplus w_1^j \oplus K1, \\ & \qquad x_1^i \oplus w_1^j \oplus K1 \oplus \Delta\} \\ 2.\ x_1^i \oplus w_1^j \oplus K1 = x_1^{i'} \oplus w_1^{j'} \oplus K_1' \oplus \Delta & \#\quad \text{(as above)} \\ 3.\ x_2^i \oplus w_1^j \oplus K1 = x_2^{i'} \oplus w_1^{j'} \oplus K_1' & \#\ u = u_2^i,\ v \in \{x_2^i \oplus w_1^j \oplus K1, \\ & \qquad x_2^i \oplus w_1^j \oplus K1 \oplus \Delta\} \\ 4.\ x_2^i \oplus w_1^j \oplus K1 = x_2^{i'} \oplus w_1^{j'} \oplus K_1' \oplus \Delta & \#\quad \text{(as above)} \end{array} \qquad (4)$$

We now analyze separately the cases where query pair i is a forward pair or a backward pair.

- If this is a forward pair then u_1^i, u_2^i are set by the adversary (with difference Δ) and x_1^i, x_2^i are drawn at random from a set of size $> 2^n - 2q$. In this case each of the equalities 1-4 from eq. (4) holds with probability at most $1/(2^n - 2q)$.
- If this is a backward pair then x_1^i is set by the adversary, u_1^i is drawn at random from a set of size $> 2^n - 2q$, u_2^i is set to $u_1^i \oplus \Delta$ and x_2^i is drawn at random from a set of size $> 2^n - 2q$. Clearly, equalities 3-4 still only hold with probability at most $1/(2^n - 2q)$ each, but we claim that the same is true also for equalities 1-2.
  To see this, recall that at this point the keys K, K' are fixed, and so are all the queries up to (but not including) the i-th pair. Denote the set of pre-images under $E_{K'}(\cdot)$ that are known so far by $U(K') = \{u_1, u_2, \ldots, u_\ell\}$ (for some ℓ), and the corresponding images by $X(K') = \{x_i = E_{K'}(u_i) : u_i \in U(K')\}$. To satisfy equality 1, the value u_1^i must be equal to some $u_k \in U(K')$. Moreover, once x_1^i is set by the adversary, there is at least one such value that would make equality 1 hold, namely the value u_k corresponding to $x_k = x_1^i \oplus w_1^j \oplus w_1^{j'} \oplus K1 \oplus K_1'$. Hence once the adversary sets x_1^i, the probability of equality 1 holding is at most $1/(2^n - 2q)$, i.e. the chance of hitting just the right value $u_1^i = u_k$. The exact same argument is true also for equality 2.

We conclude that the case where the i-th query pair completes any specific prior partial collision into a full collision only happens with probability at most $4/(2^n - 2q)$. Since key K has at most $\beta - 1$ prior partial collisions, it means that this case happens with probability at most $4(\beta - 1)/(2^n - 2q)$ which satisfies the bound in Lemma 3.

Case 2. In this case, it is a partial collision between the i-th pair itself and a prior query pair that gives us the full collision. Here we denote the i-th query by $(K = K^i, \cdot, (w_1^i, z_1^i), (w_2^i, z_2^i))$ (where $w_2^i = s_1^i \oplus \Delta$ and $\{z_1^i, z_2^i\}$ were obtained before for some other key).

We already bound the probability of any forward partial collision, here we only need to analyze the case where the i-th query pair is a backward pair. Namely the adversary sets z_1^i, then w_1^i is drawn from a set of size $> 2^n - 2q$, w_2^i is set as $w_2^i = w_1^i \oplus \Delta$ and z_2^i is drawn from a set of size $\geq 2^n - 2q$. We want to bound the probability of the event where:

- The $z_\star^i$'s collide with some prior query pair i', denoted $(K' = K^{i'}, \cdot, (w_1^{i'}, z_1^{i'}), (w_2^{i'}, z_2^{i'}))$. Namely $\{z_1^i, z_2^i\} = \{z_1^{i'}, z_2^{i'}\}$ as sets irrespective of order, which happens with probability at most $i/(2^n - 2q)$.
- In addition, there were two other query pairs involving K and K', denoted $(K = K^j, \cdot, (u_1^j, x_1^j), (u_2^j, x_2^j))$ and $(K' = K^{j'}, \cdot, (u_1^{j'}, x_1^{j'}), (u_2^{j'}, x_2^{j'}))$ (where j, j' can be the same as or different from i, i'), such that at least one of the following equalities hold:

  1. $x_1^j \oplus w_1^i \oplus K1 = x_1^{j'} \oplus w_1^{i'} \oplus K_1'$ # $u = u_1^j$, $v \in \{x_1^j \oplus w_1^j \oplus K1, x_1^j \oplus w_1^i \oplus K1 \oplus \Delta\}$
  2. $x_1^j \oplus w_1^i \oplus K1 = x_1^{j'} \oplus w_1^{i'} \oplus K_1' \oplus \Delta$ # (as above)
  3. $x_2^j \oplus w_1^i \oplus K1 = x_2^{j'} \oplus w_1^{i'} \oplus K_1'$ # $u = u_2^j$, $v \in \{x_2^j \oplus w_1^i \oplus K1, x_2^j \oplus w_1^i \oplus K1 \oplus \Delta\}$
  4. $x_2^j \oplus w_1^i \oplus K1 = x_2^{j'} \oplus w_1^{i'} \oplus K_1' \oplus \Delta$ # (as above)

  (These are exactly the same equalities as in eq. (4), except with $i \leftrightarrow j$ and $i' \leftrightarrow j'$.)

The partial collision happens with probability at most $i/(2^n - 2q)$, and since w_1^i is drawn at random then each of the equalities above holds with probability at most $1/(2^n - 2q)$, so this case only happens with probability at most $4i/(2^n - 2q)^2$.

By the above, the probability of the i-th query inducing the first full collision (when $\mathsf{maxCol} < \beta$ and no forward partial collisions) is bounded by $4(\beta - 1)/(2^n - 2q) + 4i/(2^n - 2q)^2$. Therefore, $\Pr[\text{any full collision with } \mathsf{maxCol}_\Pi < \beta \text{ and no forward partial collisions}] \leq 4(\beta - 1)q/(2^n - 2q) + \binom{q}{2}/(2^n - 2q)^2$, as needed by Lemma 3.

6 Performance

XAES introduces three additional AES256 calls to AES-GCM and KC-XAES introduces six. For comparison, DNDK [16], another efficient derived key mode, adds six AES256 calls without key commitment and ten with key commitment. DNDK version 2 [17,18], on the other hand, adds three and five, respectively. Note that DNDK v2 requires more frequent rekeys to keep the K_C collision probability conservatively low than every 2^{80} messages mandated by (KC-)XAES. Regardless, the impact of a few extra AES256 calls per invocation is minimal especially when XAES encrypts large plaintexts. To investigate this impact, we implemented XAES and KC-XAES and benchmarked them against AES-GCM in `x86_64` and `AArch64`-based platforms. Note that our implementation could be further sped up by taking advantage of vectorization and optimizing the AES256 calls (by parallelizing $\mathrm{AES256}_K(0^{128})$ with $\mathrm{AES256}_K(\mathrm{const}_3 \parallel U)$ and $\mathrm{AES256}_K(M_1)$ with $\mathrm{AES256}_K(M_2)$, $\mathrm{AES256}_K(W_1)$, and $\mathrm{AES256}_K(W_2)$ from Figure 2).

		Intel® Xeon®Platinum 8375C			AWS Graviton4 - Neoverse V2		
		AES-GCM	XAES	KC-XAES	AES-GCM	XAES	KC-XAES
Init		0.134	0.053	0.054	0.082	0.041	0.042
Encrypt	32B	0.087	0.244	0.296	0.089	0.209	0.251
	1KB	0.219	0.376	0.432	0.250	0.411	0.454
	16KB	1.448	1.620	1.686	2.644	2.767	2.810
	1MB	91.24	90.74	90.13	164.2	164.4	164.5
Decrypt	32B	0.094	0.242	0.298	0.096	0.217	0.258
	1KB	0.225	0.373	0.431	0.254	0.415	0.457
	16KB	1.457	1.618	1.671	2.587	2.711	2.752
	1MB	85.40	86.33	85.93	160.4	160.3	160.6

Table 1. AES-GCM, XAES, and KC-XAES cost in μs/operation with small and large plaintexts in `x86_64` and `AArch64`-based platforms.

Table 1 shows the performance of each mode in microseconds spent per operation in `x86_64`-based Intel® Xeon®Platinum 8375C and `AArch64`-based AWS Graviton4 (Neoverse V2) processors, respectively. The `Init` operation for AES-GCM includes a) the key scheduling, b) one AES256 encryption of a counter (for the AES-GCM tag T) and c) a pre-computation of the GHASH constants which depend on the implementation; the more blocks processed in parallel (i.e. the more optimized the implementation), the more constants are pre-computed. In the case of (KC-)XAES, `Init` includes only the AES256 encryption to produce $K1$ used in key derivation (and commitment). Note that (KC-)XAES `Encrypt` and `Decrypt` include the AES256 calls to derive the key (two calls) and the commitment string (three calls). They also include a), b) and c) for the newly-derived K_U as (KC-)XAES initializes GCM with every invocation. We can see that for small plaintexts, XAES and KC-XAES encryption and decryption cost $\times 2 - \times 3$ than AES-GCM respectively because of the cost of deriving K_U and K_C and initializing GCM for K_U. As the plaintext size increases this cost is amortized over the cost of plaintext encryption and it ends up being unnoticeable for large 1MB plaintexts. The difference between AES-GCM and (KC-)XAES is almost constant because the overhead for the key derivation and commitment does not depend on the size of the plaintext.

Table 2 shows the percentage overhead for each mode in both the Intel® and AWS Graviton4 platforms. `Init` is ~60% slower for GCM since (KC-)XAES only includes the $K1$ derivation. `Encrypt` and `Decrypt` starts much slower for (KC-)XAES for small plaintexts and drops as the cost of the GCM initialization and K_U, K_C derivation for every invocation of (KC-)XAES becomes insignificant compared to the cost of encryption/decryption of the plaintext.

Table 3 compares the schemes for small plaintexts in an Intel® Xeon®Platinum 8375C. `Init` remains cheaper for (KC-)XAES. Encryption and decryption is still expensive for (KC-)XAES especially since the plaintexts are small. Note that the amortization of the (KC-)XAES overhead compared to AES-GCM is not linear because GCM's performance also increases as the plaintext grows in size.

		Intel® Xeon® Platinum 8375C		AWS Graviton4 - Neoverse V2	
		XAES vs GCM	KC-XAES vs GCM	XAES vs GCM	KC-XAES vs GCM
Init		-60.45%	-59.70%	-50.00%	-48.78%
Encrypt	32B	180.46%	240.23%	134.83%	182.02%
	1KB	71.69%	97.26%	64.40%	81.60%
	16KB	11.88%	16.44%	4.65%	6.28%
	1MB	-0.55%	-1.22%	0.12%	0.18%
Decrypt	32B	157.45%	217.02%	126.04%	168.75%
	1KB	65.78%	91.56%	63.39%	79.92%
	16KB	11.05%	14.69%	4.79%	6.38%
	1MB	1.09%	0.62%	-0.06%	0.12%

Table 2. (KC-)XAES impact comparison against AES-GCM with small and large plaintexts in `x86_64`-based and `AArch64`-based platforms.

		AES-GCM	XAES		KC-XAES	
		μs/op	μs/op	vs GCM	μs/op	vs GCM
Init		0.134	0.054	-60.29%	0.055	-59.56%
Encrypt	32B	0.087	0.247	183.91%	0.303	248.28%
	128B	0.096	0.258	168.75%	0.313	226.04%
	256B	0.123	0.285	131.71%	0.341	177.24%
	512B	0.176	0.333	89.20%	0.388	120.45%
	1KB	0.220	0.377	71.36%	0.429	95.00%
Decrypt	32B	0.092	0.250	171.74%	0.303	229.35%
	128B	0.096	0.252	162.50%	0.306	218.75%
	256B	0.116	0.269	131.90%	0.326	181.03%
	512B	0.180	0.334	85.56%	0.388	115.56%
	1KB	0.224	0.379	69.20%	0.433	93.30%

Table 3. AES-GCM, XAES, and KC-XAES cost with small plaintexts in Intel® Xeon® Platinum 8375C.

To confirm that blockcipher-based key derivation is more performant that HMAC-based key derivation, we also compared (KC-)XAES against HMAC-based derived key modes. AES256 is generally one order of magnitude faster than HMAC in common implementations. An HMAC derived key mode is described in [7] with a goal to prevent frequent re-keying in high-scale cloud environments. HMAC-based key derivation and key commitment implementations in our experiments included the exact same operations in all `Init`, `Encrypt` and `Decrypt` operations of (KC-)XAES with the only difference being the PRF function. Table 4 shows the experimental results. We can see that when HMAC-SHA256 is used as the PRF instead of CMAC-AES256 for (KC-)XAES key derivation and key commitment, both the `Init` and `Encrypt`/`Decrypt` operations for messages up to 16KB introduce more overhead to GCM than the overhead of (KC-)XAES (Table 2). When the plaintext increases to 1MB, the cost of AES-GCM encryption and decryption diminishes the performance benefit of cheaper key derivation and commitment. Using other HMAC-based PRF functions like HKDF instead of HMAC would be of similar or worse performance.

		AES-GCM	H-AES-GCM		H-KC-AES-GCM	
		μs/op	μs/op	vs GCM	μs/op	vs GCM
		Intel® Xeon®Platinum 8375C				
Init		0.136	0.142	4%	0.142	4%
Encrypt	32B	0.086	0.357	314%	0.501	481%
	1KB	0.219	0.487	122%	0.637	191%
	16KB	1.462	1.718	17%	1.882	29%
	1MB	87.23	86.64	-1%	88.11	1%
Decrypt	32B	0.093	0.361	289%	0.506	445%
	1KB	0.225	0.493	119%	0.640	184%
	16KB	1.458	1.726	18%	1.879	29%
	1MB	85.33	85.22	0%	85.10	1%
		AWS Graviton4 - Neoverse V2				
Init		0.083	0.091	10%	0.091	10%
Encrypt	32B	0.089	0.285	220%	0.395	344%
	1KB	0.250	0.485	94%	0.594	138%
	16KB	2.643	2.841	7%	2.951	12%
	1MB	164.6	164.3	0%	164.8	0%
Decrypt	32B	0.096	0.290	202%	0.399	316%
	1KB	0.254	0.487	92%	0.595	134%
	16KB	2.587	2.784	8%	2.892	12%
	1MB	160.8	160.4	0%	160.6	0%

Table 4. AES-GCM, HMAC-based Derived Key AES-GCM (H-AES-GCM), and HMAC-based Derived Key and Key Committing AES-GCM (H-KC-AES-GCM) cost with small and large plaintexts on `x86_64` and `AArch64` processors. H-(KC-)XAES is shown much more expensive than CMAC-based (KC-)XAES in Table 2.

7 Conclusion

In conclusion, we revisited Valsorda's XAES [14] and described how to add key commitment to it, resulting in a scheme that we call KC-XAES. We showed that in the ideal-cipher model, KC-XAES provides n-bit security against the FOR-KC key commitment attacks of Farshim et al. [12]. We established usage- and data-bounds for (KC-)XAES, and argued that these modes are FIPS-compliant. We confirmed XAES is very performant, especially for larger plaintexts where the additional key derivation and/or key commitment are amortized over the encryption of the whole message. Finally, we briefly discussed applications where KC-XAES would be suitable.

8 Acknowledgements

The authors would like to thank Fabrice Benhamouda and Hugo Krawczyk from AWS for their valuable input and help.

References

1. A. Albertini, T. Duong, S. Gueron, S. Kölbl, A. Luykx, and S. Schmieg. How to abuse and fix authenticated encryption without key commitment. In K. R. B.

Butler and K. Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 3291–3308. USENIX Association, 2022.

2. Anonymized for submission. Blockcipher-Based Key Derivation without PRP/PRF Switching. Personal communications.
3. S. Arciszewski. AES with Galois Extended Mode (AES-GEM). NIST Workshop on the Requirements for an Accordion Cipher Mode, June 2024. `https://csrc.nist.gov/csrc/media/Events/2024/accordion-cipher-mode-workshop-2024/documents/papers/galois-extended-mode.pdf`.
4. M. Bellare and V. T. Hoang. Efficient schemes for committing authenticated encryption. In O. Dunkelman and S. Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022*, pages 845–875, Cham, 2022. Springer International Publishing.
5. M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In T. Okamoto, editor, *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*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000.
6. R. Bhaumik and J. P. Degabriele. Pencil: A domain-extended PRF with full n-bit security for strengthening GCM and more. Cryptology ePrint Archive, Paper 2025/383, 2025.
7. M. Campagna and S. Gueron. Key management systems at the cloud scale. *Cryptography*, 3(3), 2019.
8. C. Celi. ACVP SP800-108 Key Derivation Function JSON Specification. nist.gov, Aug. 2024. `https://pages.nist.gov/ACVP/draft-celi-acvp-kbkdf.html`.
9. L. Chen. Recommendation for Key Derivation Using Pseudorandom Functions. NIST Special Publication 800-108 Rev1 Upd1, 2024. `https://doi.org/10.6028/NIST.SP.800-108r1-upd1`.
10. Common Criteria Recognition Arrangement (CCRA) Members. Common Criteria for Information Technology Security Evaluation. commoncriteriaportal.org, June 2024. `https://www.commoncriteriaportal.org/index.cfm`.
11. M. Dworkin. Recommendation for block cipher modes of operation: Galois/counter mode (GCM) and GMAC. Technical Report NIST Special Publication (SP) 800-38D (Final), November 28, 2007, U.S. Department of Commerce, Washington, D.C., 2007.
12. P. Farshim, C. Orlandi, and R. Rosie. Security of symmetric primitives under incorrect usage of keys. *IACR Trans. Symmetric Cryptol.*, 2017(1):449–473, 2017.
13. Filippo Valsorda. XAES-256-GCM. Personal Blog, July 2024. `https://words.filippo.io/dispatches/xaes-256-gcm/`.
14. Filippo Valsorda. XAES-256-GCM Specification. GitHub, June 2024. `https://github.com/C2SP/C2SP/blob/main/XAES-256-GCM.md`.
15. S. Gueron. Key committing AEADs. Cryptology ePrint Archive, Paper 2020/1153, 2020. `https://eprint.iacr.org/2020/1153`.
16. S. Gueron. Double Nonce Derive Key AES-GCM (DNDK-GCM). Internet-Draft draft-gueron-cfrg-dndkgcm-01, Internet Engineering Task Force, Oct. 2024. Work in Progress.
17. S. Gueron. Double Nonce Derive Key AES-GCM (DNDK-GCM). Internet-Draft draft-gueron-cfrg-dndkgcm-02, Internet Engineering Task Force, Mar. 2025. Work in Progress.

18. S. Gueron and T. Ristenpart. DNDK: Combining nonce and key derivation for fast and scalable AEAD. Cryptology ePrint Archive, Paper 2025/785, 2025.
19. V. T. Hoang, S. Tessaro, and A. Thiruvengadam. The multi-user security of gcm, revisited: Tight bounds for nonce randomization. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1429–1440, New York, NY, USA, 2018. Association for Computing Machinery.
20. T. Iwata, K. Ohashi, and K. Minematsu. Breaking and repairing GCM security proofs. Cryptology ePrint Archive, Paper 2012/438, 2012.
21. P. Kampanakis, E. Crocket, M. Campagna, A. Petcher, and S. Gueron. Practical Challenges with AES-GCM and the need for a new cipher. NIST 3rd Workshop on Block Cipher Modes of Operation, Nov. 2023. `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`.
22. J. Len, P. Grubbs, and T. Ristenpart. Partitioning oracle attacks. In M. D. Bailey and R. Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 195–212. USENIX Association, 2021.
23. A. Luykx and K. G. Paterson. Limits on authenticated encryption use in TLS. Cryptology ePrint Archive, Paper 2024/051, 2024.
24. S. Menda, J. Len, P. Grubbs, and T. Ristenpart. Context discovery and commitment attacks. In C. Hazay and M. Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 379–407, Cham, 2023. Springer Nature Switzerland.
25. National Institute of Standards and Technology. Security Requirements for Cryptographic Modules. csrc.nist.gov, Mar. 2019. `https://csrc.nist.gov/pubs/fips/140-3/final`.
26. NIST. NIST Workshop on the Requirements for an Accordion Cipher Mode 2024. Workshop Report. NIST NIST Interagency Report, Apr. 2024. `https://nvlpubs.nist.gov/nistpubs/ir/2024/NIST.IR.8537.pdf`.
27. NIST. Pre-Draft Call for Comments: GCM and GMAC Block Cipher Modes of Operation. NIST SP 800-38D Rev. 1 (Initial Preliminary Draft), Apr. 2024. `https://csrc.nist.gov/pubs/sp/800/38/d/r1/iprd`.
28. NIST. PRE-DRAFT Call for Comments: NIST Proposes to Standardize a Wider Variant of AES. NIST SP 800-197 (Initial Preliminary Draft), Apr. 2024. `https://csrc.nist.gov/pubs/sp/800/197/iprd`.
29. R. Takeuchi, Y. Todo, and T. Iwata. Key recovery, universal forgery, and committing attacks against revised rocca: How finalization affects security. *IACR Trans. Symmetric Cryptol.*, 2024(2):85–117, 2024.
30. A. Tribble. Improved client-side encryption: Explicit KeyIds and key commitment. AWS Security Blog, 2020. `https://aws.amazon.com/blogs/security/improved-client-side-encryption-explicit-keyids-and-key-commitment`.

A KC-XAES Specification Breakdown

Specification 1 below breaks down the code in Figure 2 to its AES256 calls to eliminate three unnecessary subkey $K1$ and one subciphertext X_1 calculations in the CMAC-AES256 calls.

As shown in Section 4.3, an implementation of Specification 1 could be FIPS certified. To do that, someone would need to certify the underlying AES-GCM implementation, the nonce N generation method, and the K_U derivation (which can be certified as a KBKDF [9]). For a KBKDF FIPS validation of the K_U derivation, the ACVP "KDF"//"1.0" module test [8] will include

```
"kdfMode":"counter",
"macMode":"CMAC-AES256",
"counterLocation":"before fixed data",
"counterLength":16,
```

For the K_C derivation, the ACVP "KDF"//"1.0" module test will include

```
"kdfMode":"counter",
"macMode":"CMAC-AES256",
"counterLocation":"after fixed data",
"counterLength":8,
```

The module registration for both the K_U and K_C derivation will include

```
"supportedLengths":[ {"min":256, "max":256, "increment":1 } ]
```

The fixed data field in the ACVP test vector response will be

```
"fixedData":"0x5800⟨U⟩"
```

and

```
"fixedData":"0x58434D54⟨U⟩⟨V⟩000100"
```

for the K_U and K_C derivation respectively.

Specification 1 KC-XAES with a 24-byte random nonce and a 256-bit key

```
Input: $K[0:32]$, $N[0:24]$, $P$, $AAD$
Output: $C$, $T$, $K_C[0:32]$
 1: $T1 = \text{AES256}_K(0\text{x}(00)^{16})$.
 2: if $MSB_1(T1) = 0$ then
 3:     $K1[0:16] = T1 \ll 1$
 4: else
 5:     $K1[0:16] = (T1 \ll 1) \oplus 0\text{x}(00)^{15}87$        ▷ $X \times AES_K(0^{128})$
 6: end if
 7: $M_1[0:16] \leftarrow (0\text{x}00015800 \parallel N[0:12]) \oplus K1[0:16]$
 8: $K_U[0:16] \leftarrow \text{AES256}_K(M_1[0:16])$    ▷ $\text{CMAC-AES256}_K(0\text{x}00015800 \parallel U[:12])$
 9: $M_2[0:16] \leftarrow (0\text{x}00025800 \parallel N[0:12]) \oplus K1[0:16]$
10: $K_U[16:32] \leftarrow \text{AES256}_K(M_2[0:16])$    ▷ $\text{CMAC-AES256}_K(0\text{x}00025800 \parallel U[:12])$
11: $V[0:12] \leftarrow N[12:24]$
12: $(C, T) \leftarrow \text{AES-256-GCM}(K_U[0:32], AAD, IV = V, P)$
13: $X_1[0:16] \leftarrow \text{AES256}_K(0\text{x}58434D54 \parallel N[0:12])$
14: $W_1[0:16] \leftarrow X_1[0:16] \oplus (N[12:24] \parallel 0\text{x}00010001) \oplus K1[0:16]$
15: $K_C[0:16] \leftarrow \text{AES256}_K(W_1)$ ▷ $\text{CMAC-AES256}_K(\text{“XCMT”} \parallel U \parallel V \parallel 0\text{x}00010001)$
16: $W_2[0:16] \leftarrow X_1[0:16] \oplus (N[12:24] \parallel 0\text{x}00010002) \oplus K1[0:16]$
17: $K_C[16:32] \leftarrow \text{AES256}_K(W_2)$    ▷
    $\text{CMAC-AES256}_K(\text{“XCMT”} \parallel U \parallel V \parallel 0\text{x}00010002)$
18: $K_C \leftarrow K_C[0:16] \parallel K_C[16:32]$
```

B An extension that supports shorter nonces

Some constrained use-cases may not be able to use 24-byte nonce N or generate 24-byte entropy fast enough. Such scenarios could benefit from slightly smaller b-byte nonces. Figure 3 breaks down KC-XAES by using a b-byte nonce $N[:b]$ instead of constant 24-bytes, where $20 \leq b \leq 24$. To avoid trivial K_C collisions between nonces of different lengths under the same main key K, we redefine the two constants from Figure 2, depending on the length b of the nonce, as $\text{const}_4 = \langle 24 - b \rangle \parallel 0\text{x}010001$ and $\text{const}_5 = \langle 24 - b \rangle \parallel 0\text{x}010002$. XAES as specified in Figure 2 follows exactly Figure 3 for $b = 24$.

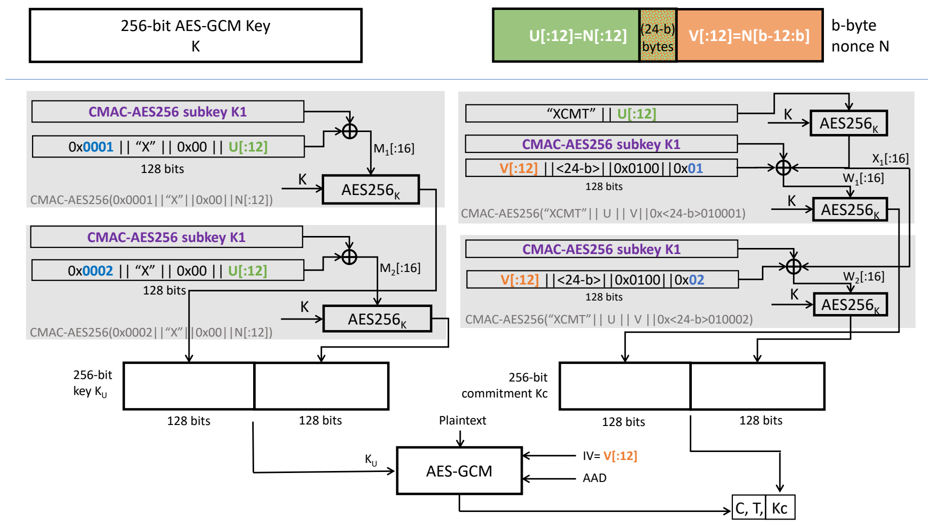

Fig. 3. KC-XAES with a b-byte random nonce N and a 256-bit key, where $20 \leq b \leq 24$

Note that when $b < 24$, $N[b - 12 : 12]$ is used as-is in the AES-GCM IV, but it passes through a pseudorandom permutation for deriving K_U. The collision-resistance analysis from section 5 applies to this setting too; using $b < 24$ does not affect the key-collision security of KC-XAES. It is worth mentioning that even in the theoretical scenario where $b = 12$, XAES is not less secure than plain AES-GCM. Additionally, K_C's derivation function remains a FIPS approved KBKDF [9, §4.1] as explained in Section 4.3 with the fixed input data becoming "XCMT"$\parallel U \parallel V \parallel \langle 24 - b \rangle \parallel 0\text{x}0100$.

Also, it is easy to adjust the data limit analysis from Section 4.2 to this case. For example, for $b = 20$ we need to bound the number of messages encrypted under one main key to at most 2^{64} to ensure a nonce collision probability of 2^{-33}. Setting the K_U re-occurrence bound $m = 4$, we get $\Pr[\text{4-collisions}] \leq \binom{2^{64}}{4} \cdot 2^{-96 \cdot 3} \approx 2^{64 \cdot 4 - 96 \cdot 3}/4! \approx 2^{-36.6} < 2^{-32}$. We can therefore assume that there will be no four-way K_U collisions for the life of the main key K, which means that every data key K_U will be used to encrypt at most three messages. The data bound per message will then be $\min(X/3, 2^{36})$, where X is the use-case specific X-bytes-per-GCM-key limit as defined in Section 4.2.

C Other Key Commitment Options

Additionally to the commitment scheme proposed in this work, we considered other constructions, some of which had already been proposed in the literature.

C.1 Padding Fix

Since XAES uses AES-GCM for encryption, then perhaps the simplest way to add key-commitment to it is using the GCM-specific padding-fix mechanism from [1]. Namely, instead of computing just AES-GCM$_{K_U}(V; P)$, prepend ℓ zero bits to the plaintext and compute AES-GCM$_{K_U}(V; 0^\ell|P)$. Albertini et al. proved in [1, Thm 1] that this yields $\ell/2$ bits of security, in the sense of FOR-KC. In particular, to get 128-bit security we need to prepend two 128-bit blocks to the plaintext before encrypting it. Decryption needs to verify that the plaintext begins with ℓ zeros, then remove those zeros and return the rest of the plaintext.

In terms of performance, this proposal adds two blocks to the ciphertext, and uses only two additional AES block encryption and two GF 128-bit block multiplication operations. The practical disadvantages of padding key commitment is its lack of modularity and the requirement for the application layer to be aware of the encryption layer which violates common programming abstractions. It also needs for decryption to take place before the commitment string can be verified to confirm that the ciphertext was produced with the expected key.

C.2 HKDF

The AWS ESDK and KMS have introduced key commitment by using HKDF with a random 256-bit `salt` and an `info` string to generate a 32-byte commitment string.

$$K_C = HKDF_K(\mathsf{salt}, ''\mathsf{COMMITKEY}'', 32)$$

This is PRF-based key commitment as described in [15]. It achieves CMT-2 security at the cost of the PRF performance. Given that CMAC-AES-based PRFs are much more performant than HKDF, we chose to pursue the former in this work.

C.3 Using less bits for K_C derivation

We considered further optimizing the performance of KC-XAES by using part of the nonce N to produce the key commitment string. For example, we could use 120-bits ($N[:15]$) with an one-byte counter which would allow for K_C to be generated with two AES256 calls instead of three.

$$\begin{aligned} K_C &= \text{CMAC-AES256}_K(0\text{x}01 \| N[:15]) \;\|\; \text{CMAC-AES256}_K(0\text{x}02 \| N[:15]) \\ &= \text{AES256}_K((0\text{x}01 \| N[:15]) \oplus K1) \;\|\; \text{AES256}_K((0\text{x}02 \| N[:15]) \oplus K1). \end{aligned}$$

Of course, that would mean we can encrypt up to 2^{48} messages before a K_C collision with 2^{-32} probability.

[17,18] follow this approach. Gueron et al. consider the collision probability and conclude that by limiting DNDK v2 to 2^{50} queries, the collision probability remains below 2^{-21} which is satisfactory as a K_C collision reveals information about the re-use

of the main key and part of the nonce, but it does not reveal information about the plaintext. Although KC-XAES could have used part of the nonce for deriving the key commitment string to save an AES256 call, we decided against it because because we considered the performance advantage marginal versus the K_C collision disadvantage.