

Security of Streaming Encryption in Google’s Tink Library

Viet Tung Hoang¹ and Yaobin Shen²

¹ Dept. of Computer Science, Florida State University

² Dept. of Computer Science & Engineering, Shanghai Jiao Tong University, China

August 23, 2020

Abstract. We analyze the multi-user security of the streaming encryption in Google’s Tink library via an extended version of the framework of nonce-based online authenticated encryption of Hoang et al. (CRYPTO’15) to support random-access decryption. We show that Tink’s design choice of using random nonces and a nonce-based key-derivation function indeed improves the concrete security bound. We then give two better alternatives that are more robust against randomness failure. In addition, we show how to efficiently instantiate the key-derivation function via AES, instead of relying on HMAC-SHA256 like the current design in Tink. To accomplish this we give a multi-user analysis of the XOR-of-permutation construction of Bellare, Krovetz, and Rogaway (EUROCRYPT’98).

1 Introduction

This paper reconsiders the problem of building nonce-based online authenticated encryption (nOAE) schemes [15]. Under this problem, one needs to encrypt potentially huge messages as soon as they arrive using constant-sized states. This kind of situation arises when Netflix sends its movie streams to users. This is what troubles memory-constrained devices when they have to deal with data that they cannot load entirely. This also happens when interactive applications, such as SSH, need to send immediately each character they receive from the keyboard.

Specifically, we need to encrypt a message $\mathbf{M}$ that is represented as a tuple $(M_1, \dots, M_m)$ of segments, with an associated data (AD) $\mathbf{A} = (A_1, \dots, A_m)$. Each time one is given a pair (M_i, A_i) to encrypt, and can only keep a small state at the end. The size of each segment can range from a character (for SSH) to a few MB (for Netflix), or even bigger. Directly encrypting each (M_i, A_i) via a conventional AE scheme such as OCB [21, 29] or GCM [24] is problematic, because an adversary can drop some of the ciphertext segments or reorder the ciphertext segments without being detected.

A GAP BETWEEN THEORY AND PRACTICE. The nOAE problem already receives a formal treatment by Hoang et al. [15], with a practical construction **STREAM** that is built on top of the OCB authenticated-encryption scheme [21, 29]. The idea is simple. Assume that at the beginning we are given a nonce N for each message $\mathbf{M}$. Assume further that we have a conventional AE scheme $\Pi = (\mathbf{K}, \mathbf{E}, \mathbf{D})$, for example one can instantiate Π as OCB. To encrypt (M_i, A_i) under key K and the base nonce N , we run $\mathbf{E}_K(N^*, A_i, M_i)$, where the derived nonce N^* is an encoding of (N, i, a) , and the flag a indicates whether M_i is the last segment, meaning that $a = 1$ if $i = m$, and $a = 0$ otherwise. See Fig. 6 for the code and an illustration of **STREAM**.

At the first glance, the problem seems solved. However, underneath the elegant design of **STREAM** lies a troubling usability issue, which comes to light when Google developers try to include a streaming encryption scheme in the Tink library [12]. Under their design, Π is instantiated via GCM [24], the most popular conventional AE scheme. For a triple (N, i, a) that will be encoded

as a 12-byte GCM nonce, Tink uses four bytes to encode i , and one byte to encode a . This means that nonces of **STREAM**[GCM] will be only 7-byte long, and thus the only viable option is to implement them as counters. Yet there are situations when random nonces are desirable:

- Routers are booted frequently, and its counters will therefore be reset often. Implementing nonces as counters in this environment will result in lots of nonce repetition.
- Synchronizing counters among busy distributed servers might be impractical.

Google developers therefore deviate from the **STREAM** design. Tink’s streaming encryption does not take nonces from users. Instead, it picks a 16-byte random salt R and a 7-byte random nonce prefix P , and derive a subkey $L \leftarrow \text{KD}(K, R)$, where KD is a key-derivation function that will be instantiated via HMAC-SHA256. It then runs **STREAM**[GCM] with key L and “nonce” P . Such short nonces will repeat, but under different subkeys, and thus will cause no harm to security.

Given that Tink’s streaming encryption deviates from the original **STREAM** design, it does not benefit from the security proof in [15]. On the other hand, Tink’s choice of random nonces and a nonce-based key-derivation may actually *increase* security, as in the case of the AES-GCM-SIV scheme [6, 13]. The goal of our paper is to analyze the security of Tink’s streaming encryption, and find ways to improve its security and efficiency, subject to the constraints that (i) Π is still instantiated via GCM, and (ii) the scheme should support flexible choices of nonce generation, including random nonces.

SECURITY OF TINK’S STREAMING ENCRYPTION. One can view Tink’s streaming encryption as an online AE scheme **SE1**[KD, Π] where a nonce is a pair (R, P) , and Tink chooses to pick nonces at random. See Fig. 8 for the code and an illustration of **SE1**.

We analyze security of **SE1** under a *strengthened* notion of nOAE, where an adversary can attack en masse instead of targeting a specific victim. In other words, security is now in the *multi-user* setting. There is a growing recognition that cryptographic primitives should be analyzed in the multi-user setting [4, 16], and it is therefore natural to put nOAE under the multi-user lens. Moreover, our notion allows *random-access decryption*, whereas the treatment of Hoang et al. assumes that decryption is in order. Random-access decryption is needed for situations where one encrypts a huge file and later want to read just a portion of the file.

Our work confirms that **SE1** is indeed secure, provided that Π is a good conventional AE scheme and KD is a good PRF. This is true for any proper choice of nonce generation. When Π is instantiated as GCM, using the recent GCM analysis of Hoang, Tessaro, and Thiruvengadam [18], picking nonces at random does indeed improve the concrete security bound. However, this is not the only good way of generating nonces for **SE1**. One can, for example, implement the salt R as a counter, but the nonce prefix P should still be picked at random.

Still, the story here is not entirely satisfactory. First, **SE1** is not robust against randomness failure, which happens frequently [11, 14, 22]. In particular, its security would degrade if R is still random, but P is a constant string. In this work, we identify two alternative constructions **SE2** and **SE3** that offer the same concrete security bound as **SE1**, but are robust to randomness failure, meaning that they only need nonces to be unique. Next, Tink’s streaming encryption uses AES for GCM, and HMAC-SHA256 for KD. It is desirable to find an AES-based instantiation of KD (for both AES-128 and AES-256) to reduce the code complexity, and improve the speed of KD.

AN AES-BASED KD. Let $E : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a blockcipher with $k \in \{n, 2n\}$ (which we would instantiate directly via AES). Our goal is to give a construction of KD on top of E that offers n -bit *multi-user* PRF security. To achieve this concrete security bound, KD should have $2n$ -bit key length to resist attacks like Biham’s key-collision attack on DES [5]. We now elaborate on how to build KD, step by step.

Step 1: Our first step is to build a blockcipher $G : \{0, 1\}^{2n} \times \{0, 1\}^n \rightarrow \{0, 1\}^n$. If $k = 2n$ then we can directly use E as our G . If $k = n$ then we extend the key length of E . In particular, for a key $K = J \parallel L$, with $|J| = |L| = n$, let

$$G_K(X) = E_J(X \oplus L) \oplus L .$$

This is a variant of Rivest’s classic DESX construction. The design of DESX uses different keys in pre-whitening and post-whitening E , whereas the variant above, which we call DESX1, uses the same key. This is now folklore knowledge that DESX1 offers n -bit multi-user PRP security. However, to the best of our knowledge, there is no explicit proof in the literature. In fact, Kilian and Rogaway [20] are the first to study the security of DESX and DESX1, but they only consider the single-user setting. Hoang and Tessaro [16] investigate the security of the *xor-cascade* construction, a multi-round generalization of DESX. They give a tight multi-user security bound for the xor-cascade, and claim that their technique can be recast to work for the multi-round generalization of DESX1, but there is no explicitly proof in their paper. Bellare and Tackmann [4] directly consider the multi-user security of DESX1, but they can only show that DESX1 has $2n/3$ -bit security.

For the sake of completeness, in this work, we give a proof in the ideal-cipher model to confirm that DESX1 indeed has n -bit multi-user PRP security.

Step 2: Our next step is to build a PRF $F : \mathcal{K} \times \{0, 1\}^{n-1} \rightarrow \{0, 1\}^n$. In particular,

$$F_K(X) = G_K(X \parallel 0) \oplus G_K(X \parallel 1) .$$

This is the XOR-or-permutation construction of Bellare, Krovetz, and Rogaway [2], which has a long line of work for just the *single-user* security. Lucks [23] proves that it can withstand to nearly $q = 2^{2n/3}$ queries. Bellare and Impagliazzo [1] give an asymptotic bound $O(n) \cdot q/2^n$, but they provide only a proof sketch. Patarin [28] claims the optimal bound $q/2^n$ via an approach that he refers to as “mirror theory”, but his proof is extremely complex with several unverifiable gaps [10, 19]. Dai, Hoang, and Tessaro [10] finally find a relatively simple proof for a bound $q/2^n + 3q^{1.5}/2^{1.5n}$ via their Chi-Squared method.

In the multi-user setting, a recent paper by Cogliati [9] claims the optimal bound $q/2^n$, but his proof relies on Patarin’s (unverifiable) mirror theory. Here we use the Chi-Squared method and Azuma’s inequality for a bound $8\sqrt{n} \cdot q/2^n$. It is a theoretically interesting open question to improve our bound to $q/2^n$ (with a simple, verifiable proof).

Step 3: Our KD only takes 15-byte inputs, namely the salt length of Tink’s streaming encryption has to be shortened a bit, but this does not affect its concrete security bound. For a string x , let $x \parallel 0^*$ denote the string obtained by padding 0’s to x until the total bit length is $n - 1$. If KD needs to output an n -bit key then

$$\text{KD}(K, X) = F_K(X \parallel 0^*) .$$

If KD needs to output a $2n$ -bit key then

$$\text{KD}(K, X) = F_K(X \parallel 0 \parallel 0^*) \parallel F_K(X \parallel 1 \parallel 0^*) .$$

Thus our instantiation for KD needs 2–4 parallelizable AES calls, which is much faster than HMAC-SHA256.

A BETTER NOAE SCHEME: SE2. To resist randomness failure in nonce generation, the most natural approach is to feed the entire nonce N to KD to generate a subkey L and a nonce prefix P . Note that here P is a part of KD’s output, instead of a part of the nonce. See Fig. 11 for the code and an illustration of **SE2**. We show that **SE2** with any proper nonce-generation mechanism has the same security as **SE1** with random nonces.

For applications that can tolerate 15-byte nonces, one can instantiate KD via the XOR construction. Depending on the subkey size, it will need 2 or 3 calls to the XOR construction, meaning 4–6 parallelizable AES calls. This is on par with the key-derivation function of AES-GCM-SIV [13].

For applications that demand long nonces, it is unclear how to find a good AES-based instantiation of KD, which we leave as an open problem. We instead propose another nOAE scheme **SE3** that has the best of both worlds: (i) its nOAE security is as good as **SE2**, and (ii) it supports long nonces, say 22 bytes, and (iii) its key-derivation can be built via the XOR construction.

AN EVEN BETTER NOAE SCHEME: SE3. In **SE3**, we treat a nonce N as a pair (R, P) as in **SE1**. We then run $\text{KD}(K, R)$ to derive a subkey L and a mask X to whiten P , obtaining an effective nonce prefix $P^* \leftarrow P \oplus X$. Nonces for GCM are based on P^* instead.

We show that **SE3** enjoys the same security as **SE2**. Moreover, it is possible to instantiate KD via the XOR construction, provided that the salt length is 15 bytes, and the nonce prefix length is still 7 bytes. Depending on the subkey size, KD will need 2 or 3 calls to the XOR construction, getting the same speed as the key-derivation function of AES-GCM-SIV.

RECOMMENDATIONS. Based on the analysis above, we recommend Tink developers to use **SE3** with 256-bit key length and instantiate KD via the XOR construction. Note that using 256-bit key still allows one to use AES-128 for both KD (via the DESX1 paradigm) and GCM (by deriving a 128-bit subkey L).

2 Preliminaries

NOTATION. For a finite set S , we let $x \leftarrow_s S$ denote the uniform sampling from S and assigning the value to x . Let $|x|$ denote the length of the string x , and for $1 \leq i < j \leq |x|$, let $x[i : j]$ denote the substring from the i th bit to the j th bit (inclusive) of x . For a vector $\mathbf{X}$ of strings, we let $|\mathbf{X}|$ denote the number of components, and $\mathbf{X}[i]$ be the i -th component. Indexing starts from 1, and thus $\mathbf{X} = (\mathbf{X}[1], \dots, \mathbf{X}[m])$, where $m = |\mathbf{X}|$. If A is a probabilistic algorithm, we let $y \leftarrow_s A(x_1, \dots)$ denote running A on inputs $x_1, \dots$ with truly random coins and assigning the output to y . For a domain Dom and a range Rng , we let $\text{Func}(\text{Dom}, \text{Rng})$ denote the set of all functions $f : \text{Dom} \rightarrow \text{Rng}$. Let $\text{Perm}(n)$ denote the set of all permutations on $\{0, 1\}^n$.

GAMES. We use the game-playing framework of [3]. (See Fig. 1 for an example.) A game begins with a procedure **INITIALIZE**, ends with a procedure **FINALIZE**, and has named oracles. When the adversary terminates with an output, we will feed this output to **FINALIZE**. We write $G(\mathcal{A}) \Rightarrow b$ to denote the event of running game G with an adversary $\mathcal{A}$ that results in b by **FINALIZE**. We also write $G(\mathcal{A})$ to abbreviate $G(\mathcal{A}) \Rightarrow \text{true}$.

procedure INITIALIZE	procedure EVAL(i, M)
$K_1, K_2, \dots, \leftarrow \$ \mathcal{K}; b \leftarrow \$ \{0, 1\}$	$Y_1 \leftarrow F(K_i, M); Y_0 \leftarrow f_i(M)$
$f_1, f_2, \dots, \leftarrow \$ \text{Func}(\mathcal{M}, \{0, 1\}^n)$	return Y_b
procedure FINALIZE(b')	
return ($b' = b$)	

Fig. 1: Game $\mathbf{G}_F^{\text{prf}}$ defining multi-user prf security of a function F .

procedure INITIALIZE	procedure EVAL(i, M)
$K_1, K_2, \dots, \leftarrow \$ \mathcal{K}; b \leftarrow \$ \{0, 1\}$	$Y_1 \leftarrow E(K_i, M); Y_0 \leftarrow \pi_i(M)$
$\pi_1, \pi_2, \dots, \leftarrow \$ \text{Perm}(n)$	return Y_b
procedure FINALIZE(b')	
return ($b' = b$)	

Fig. 2: Game $\mathbf{G}_E^{\text{prp}}$ defining multi-user prp security of a blockcipher E .

SYSTEMS AND TRANSCRIPTS. Following the notation from [16], it is convenient to consider interactions of a distinguisher $\mathcal{A}$ with an abstract system $\mathbf{S}$ which answers $\mathcal{A}$'s queries. The resulting interaction then generates a transcript $\tau = ((X_1, Y_1), \dots, (X_q, Y_q))$ of query-answer pairs. It is known that $\mathbf{S}$ is entirely described by the probabilities $\mathbf{ps}(\tau)$ that correspond to the system $\mathbf{S}$ responding with answers as indicated by τ when the queries in τ are made.

We will generally describe systems informally, or more formally in terms of a set of oracles they provide, and only use the fact that they define corresponding probabilities $\mathbf{ps}(\tau)$ without explicitly giving these probabilities.

MULTI-USER PRF. Let $F : \mathcal{K} \times \mathcal{M} \rightarrow \{0, 1\}^n$ be a function. For an adversary $\mathcal{A}$, let

$$\text{Adv}_F^{\text{prf}}(\mathcal{A}) = 2 \Pr[\mathbf{G}_F^{\text{prf}}(\mathcal{A})] - 1 ,$$

be the advantage of the adversary against the multi-user PRF security of F , where game $\mathbf{G}_F^{\text{prf}}$ is defined in Fig. 1.

For any function F of key length k , by adapting Biham's key-collision attack on DES [5], one can obtain PRF advantage $pq/2^{k+2}$ using $O(q)$ queries and $O(p)$ calls to F . See Appendix B for a description and a rigorous analysis of this attack.

MULTI-USER PRP. Let $E : \mathcal{K} \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a blockcipher. For an adversary $\mathcal{A}$, let

$$\text{Adv}_E^{\text{prp}}(\mathcal{A}) = 2 \Pr[\mathbf{G}_E^{\text{prp}}(\mathcal{A})] - 1 ,$$

be the advantage of the adversary against the multi-user PRP security of E , where game $\mathbf{G}_E^{\text{prp}}$ is defined in Fig. 2. This game is written as a standard-model notion; in the ideal-cipher model, the adversary will be given oracle access to E and its inverse as well.

3 Nonce-based Online AE

In this section, we will recall the syntax and security notion for (nonce-based) online AE of Hoang et al. [15]. We however present a single, unified security notion nOAE instead of giving separate

procedure INITIALIZE $K_1, K_2, \dots \leftarrow \mathcal{K}$ $b \leftarrow \{0, 1\}$ procedure ENC(i, N, A, M) $C_1 \leftarrow \mathbf{E}(K_i, N, A, M)$; $C_0 \leftarrow \{0, 1\}^{ C_1 }$ return C_b	procedure DEC(i, N, A, C) if $b = 0$ then return false $M \leftarrow \mathbf{D}(K_i, N, A, C)$ return ($M \neq \perp$) procedure FINALIZE(b) return ($b' = b$)
--	---

Fig. 3: Game $\mathbf{G}_H^{\text{ae}}(\mathcal{A})$ that defines the (multi-user) security of a conventional AE scheme $\Pi = (\mathbf{K}, \mathbf{E}, \mathbf{D})$.

privacy and authenticity definitions. In addition, compared to the notion of Hoang et al., ours is stronger, as (i) the adversary can target multiple users, and (ii) for each user, it is allowed to make multiple forgery attempts.

The treatment in [15] however assumes that a receiver will perform decryption in an in-order fashion, but there are applications that demand *random-access* decryption. We identify a class of online AE schemes that can support random-access decryption, formalize a corresponding (multi-user) notion nOAE2, and explore the relationship between nOAE and nOAE2.

As a warm-up, we will begin with the multi-user version of the classical security notion of conventional AE schemes.

CONVENTIONAL AE. A conventional AE scheme Π is a triple of algorithms $(\mathbf{K}, \mathbf{E}, \mathbf{D})$; the last two are deterministic. The key-generation algorithm $\mathbf{K}$ samples a key K at random. The encryption algorithm $\mathbf{E}$ takes as input a key K , a message M , a nonce N , and an associated data (AD) A to produce a ciphertext C . Given a key K , a nonce N , an AD A , and a ciphertext C , the decryption algorithm $\mathbf{D}$ either produces a message M , or outputs a symbol $\perp$ indicating invalidity. For correctness, we require that decryption reverses encryption, meaning that if $C \leftarrow \mathbf{E}(K, N, A, M)$ then $M \leftarrow \mathbf{D}(K, N, A, C)$. In this paper, we assume that there is a constant τ that a ciphertext is always τ -bit longer than its plaintext. For example, in GCM, $\tau = 128$.

Given an adversary $\mathcal{A}$, define

$$\text{Adv}_H^{\text{ae}}(\mathcal{A}) = 2 \Pr[\mathbf{G}_H^{\text{ae}}(\mathcal{A})] - 1$$

as the advantage of the adversary against the (multi-user) nonce-based security of a conventional AE scheme Π , where game $\mathbf{G}_H^{\text{ae}}$ is defined in Fig. 3. The adversary is given an encryption oracle and a decryption one. For encryption queries ENC(i, N, A, M), we require that the adversary be *nonce-respecting*, meaning that each pair (i, N) must never repeat. The adversary may repeat nonces in decryption queries, but to avoid trivial wins, if the adversary queries ENC(i, N, A, M) to receive C , subsequently it is prohibited from querying DEC(i, N, A, C). We note that the decryption oracle only tells the adversary whether a query succeeds, but does not return the decrypted message.

For any conventional AE scheme of key length k , by adapting Biham’s key-collision attack on DES [5], one can obtain advantage $pq/2^{k+2}$ by making $O(q)$ encryption queries and $O(p)$ calls to the encryption scheme. See Appendix B for a description and a rigorous analysis of this attack. Still, this attack requires that some particular nonce is used for all users, which is a realistic scenario if one implements nonces via a counter. However, if there is some variation in choosing nonces among users—for example, if one picks nonces at random—then this attack can be mitigated.

Following Bose, Hoang, and Tessaro [6], we say that an adversary is *d-repeating* if among its encryption queries, each nonce is used for at most d users. Note that we levy no restriction on nonces of decryption queries. If we restrict to d -repeating adversaries then the key-collision attack only achieves advantage $dq/2^{k+2}$.

ONLINE AE. An online AE scheme Π is a triple $(\mathcal{K}, \mathcal{E}, \mathcal{D})$. The key-generation algorithm $\mathcal{K}$, as usual, samples a key K at random. Messages, AD, and ciphertexts are vectors of strings. We speak of *segmented messages*, *segmented AD*, and *segmented ciphertexts* to refer to those vectors, and *message segments*, *AD segments*, and *ciphertext segments* when we refer to their components. Nonces however are still short strings.

The encryption scheme $\mathcal{E}$ consists of three deterministic algorithms $(\mathcal{E}.\text{init}, \mathcal{E}.\text{next}, \mathcal{E}.\text{last})$.

- Algorithm $\mathcal{E}.\text{init}$ takes as input a key K and a nonce N to produce a state S . This is used to initiate the encryption of a segmented message with a segmented AD.
- Algorithm $\mathcal{E}.\text{next}$ takes as input a state S , an AD segment A , and a message segment M , to produce a ciphertext segment C and an updated state S^* .
- Algorithm $\mathcal{E}.\text{last}$ takes as input a state, an AD segment, and a message segment, to produce a ciphertext segment. This is used for the last AD segment and message segment, closing the encryption.

In particular, to encrypt a segmented message $\mathbf{M} = (M_1, \dots, M_m)$ and a segmented AD $\mathbf{A} = (A_1, \dots, A_m)$ under a nonce N and a key K , one runs the following code, resulting in a segmented ciphertext $\mathbf{C} = (C_1, \dots, C_m)$.

```

 $S_0 \leftarrow \mathcal{E}.\text{init}(K, N)$ 
for  $i \leftarrow 1$  to  $m - 1$  do  $(C_i, S_i) \leftarrow \mathcal{E}.\text{next}(S_{i-1}, A_i, M_i)$ 
 $C_m \leftarrow \mathcal{E}.\text{last}(S_{m-1}, A_m, M_m)$ 
return  $(C_1, \dots, C_m)$ 

```

The decryption scheme $\mathcal{D}$ likewise consists of three deterministic algorithms $(\mathcal{D}.\text{init}, \mathcal{D}.\text{next}, \mathcal{D}.\text{last})$. To decrypt a segmented ciphertext $\mathbf{C} = (C_1, \dots, C_m)$ and a segmented AD $\mathbf{A} = (A_1, \dots, A_m)$ under a nonce N and a key K , one runs the following code, resulting in a (possibly partial) segmented message.

```

 $S_0 \leftarrow \mathcal{D}.\text{init}(K, N)$ 
for  $i \leftarrow 1$  to  $m - 1$  do
   $(M_i, S_i) \leftarrow \mathcal{D}.\text{next}(S_{i-1}, A_i, C_i)$ 
  if  $M_i = \perp$  then return  $(M_1, \dots, M_{i-1})$ 
 $M_m \leftarrow \mathcal{D}.\text{last}(S_{m-1}, A_m, C_m)$ 
if  $M_m = \perp$  then return  $(M_1, \dots, M_{m-1})$ 
return  $(M_1, \dots, M_m)$ 

```

Note that in the code above, if some (A_i, C_i) is invalid and consequently $M_i = \perp$, then we immediately terminate the decryption process, returning the partial segmented message $(M_1, \dots, M_{i-1})$. For correctness, we require that decryption reverses encryption.

THE NOAE NOTION. Given an adversary $\mathcal{A}$, define

$$\text{Adv}_{\Pi}^{\text{noae}}(\mathcal{A}) = 2 \Pr[\mathbf{G}_{\Pi}^{\text{noae}}(\mathcal{A})] - 1$$

<pre> procedure INITIALIZE $K_1, K_2, \dots \leftarrow \mathcal{K}; J_1, J_2, \dots \leftarrow 0; b \leftarrow \{0, 1\}$ procedure Enc.init(i, N) $J_i \leftarrow J_i + 1; j \leftarrow J_i; S_{i,j} \leftarrow \mathcal{E}.init(K_i, N); \text{return } J_i$ procedure Enc.next(i, j, A, M) if $S_{i,j} = \perp$ then return $\perp$ $(C_1, S_{i,j}) \leftarrow \mathcal{E}.next(S_{i,j}, A, M); C_0 \leftarrow \{0, 1\}^{ C_1 }$ return C_b procedure Enc.last(i, j, A, M) if $S_{i,j} = \perp$ then return $\perp$ $C_1 \leftarrow \mathcal{E}.last(S_{i,j}, A, M); C_0 \leftarrow \{0, 1\}^{ C_1 }; S_{i,j} \leftarrow \perp$ return C_b </pre>	<pre> procedure Dec($i, N, \mathbf{A}, \mathbf{C}, a$) if $b = 0$ or $\mathbf{A} \neq \mathbf{C}$ then return false $S \leftarrow \mathcal{D}.init(K_i, N); m \leftarrow \mathbf{C}$ for $j \leftarrow 1$ to $m - a$ do $(M, S) \leftarrow \mathcal{D}.next(S, \mathbf{A}[j], \mathbf{C}[j])$ if $M = \perp$ then return false return ($a = 0$ or $\mathcal{D}.last(S, \mathbf{A}[m], \mathbf{C}[m]) \neq \perp$) procedure FINALIZE(b') return ($b' = b$) </pre>
---	---

Fig. 4: **Game defining the (multi-user) nOAE security of an online AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$.**

as the advantage of the adversary against the (multi-user) nonce-based security of an online AE scheme Π , where game $\mathbf{G}_{\Pi}^{\text{nOAE}}$ is defined in Fig. 4. In this game, the adversary is given access to four oracles Enc.init, Enc.next, Enc.last, and Dec.

In Enc.init(i, N), the adversary indicates that it wants to encrypt a new segmented message for user i under nonce N , and receives an index j of the segmented message. Again we require that the adversary be *nonce-respecting*, meaning that each pair (i, N) never repeats. Each user may encrypt concurrently multiple segmented messages; when the adversary calls Enc.next or Enc.last, it has to specify the indices of the user and segmented message that it wants to attack. Once the adversary terminates the encryption of a segmented message via Enc.last, subsequent queries of Enc.next or Enc.last on that particular index will be rejected.

To use the decryption oracle, the adversary needs to provide a (possibly partial) segmented AD $\mathbf{A}$ and a segmented ciphertext $\mathbf{C}$ of the same length, and a bit a to indicate whether those are full. If the challenge bit b is 1 then the oracle indicates whether the entire $(\mathbf{A}, \mathbf{C})$ is successfully decrypted (but does not return the decrypted partial segmented message), otherwise it will always return false. To prevent a trivial win, the adversary is prohibited from the following behaviors:

- Calling $j \leftarrow \text{Enc.init}(i, N)$, and $C_k \leftarrow \text{Enc.next}(i, j, A_k, M_k)$ for $k = 1, \dots, m$, and then querying Dec($i, N, \mathbf{A}, \mathbf{C}, 0$), where $\mathbf{A} = (A_1, \dots, A_m)$ and $\mathbf{C} = (C_1, \dots, C_m)$.
- Calling $j \leftarrow \text{Enc.init}(i, N)$, and $C_k \leftarrow \text{Enc.next}(i, j, A_k, M_k)$ for $k = 1, \dots, m - 1$, and $C_m \leftarrow \text{Enc.last}(i, j, A_m, M_m)$, and then querying Dec($i, N, \mathbf{A}, \mathbf{C}, 1$), where $\mathbf{A} = (A_1, \dots, A_m)$ and $\mathbf{C} = (C_1, \dots, C_m)$.

RANDOM-ACCESS DECRYPTION. The treatment above of online AE assumes that a receiver will perform decryption in an in-order fashion. However, there are applications that demand *random-access* decryption. For example, suppose that one encrypts a huge file, where each segment is a few MB. Later, when one needs to read some portion of a file, it would be very time-consuming if one has to decrypt from the beginning to the needed location. It is therefore desirable to decrypt just the ciphertext segments containing the part of the file one wants to read.

We now identify a class of online AE scheme that can be used for random-access decryption. Specifically, we say that an online AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is *canonical* if (i) when we call

$\mathcal{D}.\text{init}(K, N)$, we get back a state of the form $(1, \sigma)$, and (ii) for each call $\mathcal{D}.\text{next}(S, A, C)$ with $S = (i, \sigma)$, the updated state S^* is $(i + 1, \sigma)$. Thus a receiver can first run $\mathcal{D}.\text{init}(K, N)$ to store the sub-state σ . Subsequently, each time when she needs to decrypt the i -th ciphertext segment C and its corresponding AD segment A , she will call $\mathcal{D}.\text{next}(S, A, C)$, where $S \leftarrow (i, \sigma)$.

THE nOAE2 NOTION. By adapting the nOAE notion, we can define an nOAE2 notion for canonical online AE schemes such that decryption queries can have random access. In particular, let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a canonical online AE scheme. For an adversary $\mathcal{A}$, define

$$\text{Adv}_{\Pi}^{\text{nOAE2}}(\mathcal{A}) = 2 \Pr[\mathbf{G}_{\Pi}^{\text{nOAE2}}(\mathcal{A})] - 1 ,$$

where game $\mathbf{G}_{\Pi}^{\text{nOAE2}}$ is defined in Fig. 5. This game is essentially the same as game $\mathbf{G}_{\Pi}^{\text{nOAE}}(\mathcal{A})$, except that for each decryption query, the adversary has to provide a list $\mathcal{I}$ of indices of the segments in $\mathbf{C}$ and $\mathbf{A}$, and the oracle only attempts to decrypt those segments. Again the adversary is required to be nonce-respecting. To prevent a trivial win, the adversary is prohibited from the following behaviors:

- Calling $j \leftarrow \text{Enc}.\text{init}(i, N)$, and $C_k \leftarrow \text{Enc}.\text{next}(i, j, A_k, M_k)$ for $k = 1, \dots, m$, and then querying $\text{Dec}(i, N, \mathbf{A}, \mathbf{C}, \mathcal{I}, 0)$ such that $\mathbf{A}[j] = A_j$ and $\mathbf{C}[j] = C_j$ for every $j \in \mathcal{I}$.
- Calling $j \leftarrow \text{Enc}.\text{init}(i, N)$, and $C_k \leftarrow \text{Enc}.\text{next}(i, j, A_k, M_k)$ for $k = 1, \dots, m$, and then querying $\text{Dec}(i, N, \mathbf{A}, \mathbf{C}, \mathcal{I}, 1)$ such that $|\mathbf{C}| \notin \mathcal{I}$ and $\mathbf{A}[j] = A_j$ and $\mathbf{C}[j] = C_j$ for every $j \in \mathcal{I}$.
- Calling $j \leftarrow \text{Enc}.\text{init}(i, N)$, and $C_k \leftarrow \text{Enc}.\text{next}(i, j, A_k, M_k)$ for $k = 1, \dots, m - 1$, and $C_m \leftarrow \text{Enc}.\text{last}(i, j, A_m, M_m)$, and then querying $\text{Dec}(i, N, \mathbf{A}, \mathbf{C}, \mathcal{I}, 1)$ such that $\mathbf{A}[j] = A_j$ and $\mathbf{C}[j] = C_j$ for every $j \in \mathcal{I}$.

We can view nOAE2 as a way to generalize nOAE by using an adversarial list $\mathcal{I}$ to allow random-access decryption instead of fixing $\mathcal{I} = (1, \dots, |\mathbf{C}|)$ for sequential decryption. In Appendix A, we show a contrived canonical online AE scheme that is nOAE-secure, but trivially broken under the nOAE2 notion. Thus for canonical online AE schemes, nOAE2 is strictly stronger than nOAE. Since all online AE schemes in this paper are canonical, we will focus on the nOAE2 notion.

4 Tink’s Streaming Encryption

In this section we describe the streaming encryption in Google’s Tink library, and analyze its security. We begin by recalling the **STREAM** paradigm of Hoang et al. [15] that Tink’s streaming encryption loosely follows.

4.1 The STREAM Construction

THE SCHEME. Hoang et al. [15] give a construction **STREAM** that turns a conventional AE scheme to an nOAE-secure canonical online AE one. In particular, let $\Pi = (\mathbf{K}, \mathbf{E}, \mathbf{D})$ be a conventional AE scheme. The construction $\mathbf{STREAM}[\Pi] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ has the same key-generation algorithm as Π , meaning that $\mathcal{K} = \mathbf{K}$. Its specification is given in Fig. 6.

SECURITY OF THE STREAM CONSTRUCTION. Hoang et al. [15] only provide a single-user analysis of the nOAE security of **STREAM**. Below, we will give a multi-user nOAE2 bound.

<pre> procedure INITIALIZE $K_1, K_2, \dots \leftarrow \mathcal{K}; J_1, J_2, \dots \leftarrow 0; b \leftarrow \{0, 1\}$ procedure Enc.init(i, N) $J_i \leftarrow J_i + 1; j \leftarrow J_i; S_{i,j} \leftarrow \mathcal{E}.init(K_i, N); \text{return } J_i$ procedure Enc.next(i, j, A, M) if $S_{i,j} = \perp$ then return $\perp$ $(C_1, S_{i,j}) \leftarrow \mathcal{E}.next(S_{i,j}, A, M); C_0 \leftarrow \{0, 1\}^{ C_1 }$ return C_b procedure Enc.last(i, j, A, M) if $S_{i,j} = \perp$ then return $\perp$ $C_1 \leftarrow \mathcal{E}.last(S_{i,j}, A, M); C_0 \leftarrow \{0, 1\}^{ C_1 }; S_{i,j} \leftarrow \perp$ return C_b </pre>	<pre> procedure Dec($i, N, \mathbf{A}, \mathbf{C}, \mathcal{I}, a$) if $b = 0$ or $\mathbf{A} \neq \mathbf{C}$ then return false $(1, \sigma) \leftarrow \mathcal{D}.init(K_i, N); m \leftarrow \mathbf{C}$ for $r \leftarrow 1$ to $\mathcal{I}$ do if $\mathcal{I}[r] > m$ or $\mathcal{I}[r] < 1$ then return false if $(a = 0 \text{ or } \mathcal{I}[r] < m)$ then $j \leftarrow \mathcal{I}[r]; S \leftarrow (j, \sigma)$ $(M, S) \leftarrow \mathcal{D}.next(S, \mathbf{A}[j], \mathbf{C}[j])$ else $S \leftarrow (m, \sigma); (M, S) \leftarrow \mathcal{D}.last(S, \mathbf{A}[m], \mathbf{C}[m])$ if $M = \perp$ then return false return true procedure FINALIZE(b') return $(b' = b)$ </pre>
---	--

Fig. 5: Game defining the (multi-user) nOAE2 security of a canonical online AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$.

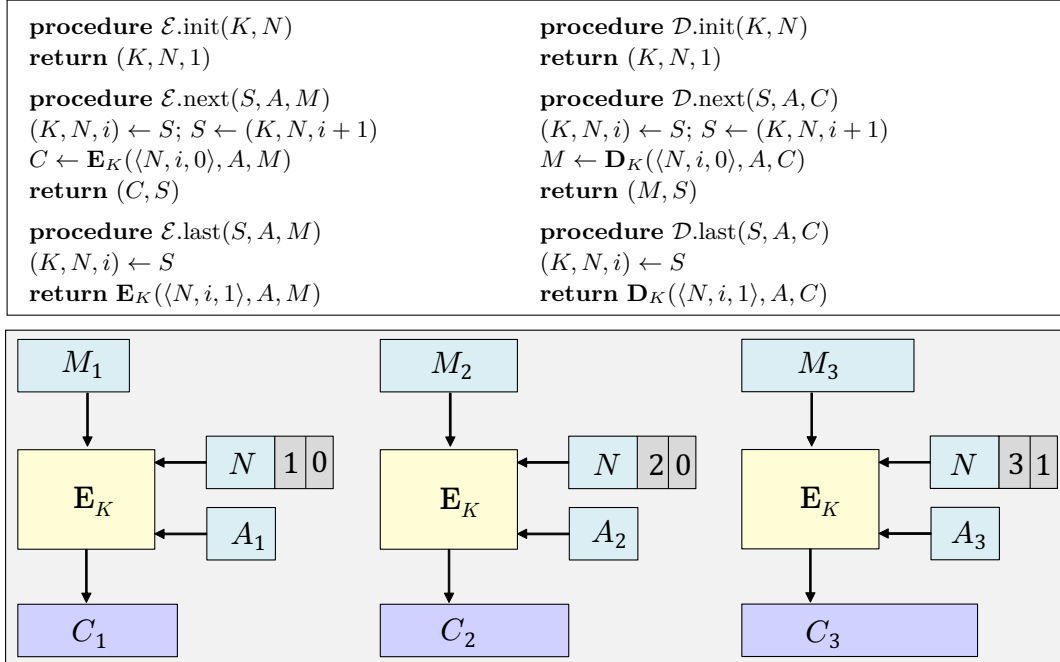


Fig. 6: The **STREAM** construction $\mathbf{STREAM}[\Pi] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, built on top of a conventional AE scheme $\Pi = (\mathbf{K}, \mathbf{E}, \mathbf{D})$. Here $\langle \cdot \rangle$ is a string encoding of a triple (N, i, a) , where N is a nonce, i is a counter, and a is a bit.

Proposition 1. *Let Π be a conventional AE scheme. For any adversary $\mathcal{A}$ attacking $\mathbf{STREAM}[\Pi]$, we can construct another adversary $\mathcal{B}$ such that*

$$\text{Adv}_{\mathbf{STREAM}[\Pi]}^{\text{noae2}}(\mathcal{A}) = \text{Adv}_{\Pi}^{\text{ae}}(\mathcal{B}) .$$

<pre> procedure Enc.init(i, N) if $J_i = \perp$ then $J_i \leftarrow 0$ $J_i \leftarrow J_i + 1$; $j \leftarrow J_i$; $S_{i,j} \leftarrow (N, 1)$; return J_i procedure Enc.next(i, j, A, M) if $S_{i,j} = \perp$ then return $\perp$ $(N, \ell) \leftarrow S_{i,j}$; $S_{i,j} \leftarrow (N, \ell + 1)$; $N^* \leftarrow \langle N, \ell, 0 \rangle$ $C \leftarrow \text{ENC}(i, N^*, A, M)$; $\text{Tab}[i, N^*, A, C] \leftarrow \text{true}$ return C procedure Enc.last(i, j, A, M) if $S_{i,j} = \perp$ then return $\perp$ $(N, \ell) \leftarrow S_{i,j}$; $S_{i,j} \leftarrow \perp$; $N^* \leftarrow \langle N, \ell, 1 \rangle$ $C \leftarrow \text{ENC}(i, N^*, A, M)$; $\text{Tab}[i, N^*, A, C] \leftarrow \text{true}$ return C </pre>	<pre> procedure Dec($i, N, \mathbf{A}, \mathbf{C}, \mathcal{I}, a$) if $\mathbf{A} \neq \mathbf{C}$ then return false for $r \leftarrow 1$ to $\mathcal{I}$ do $m \leftarrow \mathbf{C}$; $j \leftarrow \mathcal{I}[r]$ if $j > m$ or $j < 1$ then return false if $(a = 0 \text{ or } j < m)$ then if $\text{LOOK}(i, \langle N, j, 0 \rangle, \mathbf{A}[j], \mathbf{C}[j]) = \text{false}$ then return false elseif $\text{LOOK}(i, \langle N, m, 1 \rangle, \mathbf{A}[m], \mathbf{C}[m]) = \text{false}$ then return false return true private procedure LOOK(i, N^*, A, C) if $\text{Tab}[i, N^*, A, C] \neq \perp$ then return true return DEC(i, N^*, A, C) </pre>
---	---

Fig. 7: Constructed adversary $\mathcal{B}$ in the proof of Proposition 1. It is given two oracles ENC and DEC, and maintains an internal procedure LOOK.

Adversary $\mathcal{B}$ uses about the same running time and makes as many encryption queries as $\mathcal{A}$. It also makes at most as many decryption queries as the total number of ciphertext segments in $\mathcal{A}$'s decryption queries.

Proof. Given an adversary $\mathcal{A}$ attacking **STREAM** $[\Pi]$, we construct an adversary $\mathcal{B}$ attacking Π as follows. Adversary $\mathcal{B}$ runs $\mathcal{A}$ and simulates game $\text{Adv}_{\text{STREAM}[\Pi]}^{\text{noae2}}(\mathcal{A})$ with challenge bit $b = 1$, but the oracles are implemented as shown in Fig. 7. Informally, for each call $\mathbf{E}_K(i, N^*, A, M)$, adversary $\mathcal{B}$ will make the corresponding query (i, N^*, A, M) to its oracle ENC to get the answer C , and stores the tuple (i, N^*, A, C) . For each call $\mathbf{D}_K(i, N^*, A, C)$, if there is no prior tuple (i, N^*, A, C) then the adversary will query DEC(i, N^*, A, C). However, if there is already such a tuple then querying DEC is prohibited; in that case the adversary returns true instead. Since $\mathcal{A}$ is nonce-respecting and since we include a counter in creating the nonces of **STREAM**, adversary $\mathcal{B}$ is also nonce-respecting. When $\mathcal{A}$ finishes and outputs a bit b' , adversary $\mathcal{B}$ outputs the same bit.

Due to the restriction on the behavior of $\mathcal{A}$, any call to the Dec oracle of $\mathcal{A}$ must eventually end up with a call to the oracle DEC of $\mathcal{B}$ instead of looking up prior tuples from the encryption queries. As a result, if the challenge bit of game $\mathbf{G}_{\Pi}^{\text{ae}}(\mathcal{B})$ is 0 then the Dec oracle will always return false. Hence $\mathcal{B}$ perfectly simulates game $\text{Adv}_{\text{STREAM}[\Pi]}^{\text{noae2}}(\mathcal{A})$, and thus

$$\text{Adv}_{\Pi}^{\text{ae}}(\mathcal{B}) = \text{Adv}_{\text{STREAM}[\Pi]}^{\text{noae2}}(\mathcal{A})$$

as claimed.

DISCUSSION. In **STREAM** $[\Pi]$, a nonce in the conventional AE scheme Π is formed from a base nonce N of **STREAM** $[\Pi]$, a counter i (to indicate the position of the message segment within the segmented message), and a bit a (to indicate whether the segment is the last one). Because the range of the counter i limits the size of segmented messages, a practical encoding choice would be to use 4 bytes for i and one byte for a , as in Tink's implementation. This approach is fine if (i) the scheme Π supports somewhat long nonces—for example OCB [21] allows nonces to be up to 15 bytes, and (ii) one implements nonces as counters. However, the most popular conventional

AE scheme, GCM [24], dictates that its nonces must be exactly 12-byte long, leaving only 7-byte space for nonces of **STREAM**[GCM]. In addition, there are situations when it is desirable to use random nonces:

- Routers are booted frequently, and its counters will therefore be reset often. Implementing nonces as counters in this environment will result in lots of nonce repetition.
- Synchronizing counters among busy distributed servers might be impractical.

Thus the **STREAM** construction, while simple and elegant, may be unusable in some settings. In fact, the implementation in Tink library does not closely follow the **STREAM** paradigm.

4.2 Online AE in Tink Library

THE SCHEME. Tink’s streaming encryption deviates from the syntax of online AE in several ways:

- First, under Tink’s streaming encryption, there is no segmented AD. A user instead needs to provide a (possibly empty) *header* H when she initiates encryption/decryption. This header serves the same role as AD.
- Next, the API does not let users provide nonces. Instead, the encryption scheme will pick up a 7-byte *nonce prefix* P and a 16-byte *salt* S^* at random. It then uses HMAC-SHA256 to derive an internal subkey L from $R \leftarrow S^* \parallel H$, and runs **STREAM**[GCM] (without segmented AD) under the key L and nonce P . The string $S^* \parallel P$ is added to the first ciphertext segment.

At the first glance, there are several issues in Tink’s streaming encryption.

- First, as it does not even follow the syntax of an online AE scheme, it is unclear what kind of security Tink’s streaming encryption offers.
- Next, relying on true randomness (instead of uniqueness of nonces) is a step backward in robustness towards randomness failure, which is quite common [11, 14, 22].
- Finally, requiring users to provide the entire associated data at the beginning in a header H reduces the usability of the scheme.

To understand the security of Tink’s streaming encryption, we consider a generalized canonical scheme **SE1** that supports segmented AD—Tink’s streaming encryption corresponds to the special case in which the adversary only pick AD segments as the empty string. To cast **SE1** into the framework of online AE, we view the pair (R, P) , with $R = S^* \parallel H$, as a nonce. Under this viewpoint, Tink’s streaming encryption ensures the uniqueness of nonces by incorporating some contextualization information H , and at the same time also authenticates H . By only requiring a mild condition that (R, P) is unique, **SE1** is robust to randomness failure. The code of **SE1** is shown in Fig. 8. This scheme makes use of a generic key-derivation function KD and a conventional AE scheme that Tink instantiates via HMAC-SHA256 and GCM respectively.

ANALYSIS OF **SE1**. The following result shows that **SE1**[KD, Π] meets the nOAE2 notion, provided that KD is a good PRF and Π is a good conventional AE scheme. The proof is in Section 4.3.

Theorem 1. *Let $\Pi = (\mathbf{K}, \mathbf{E}, \mathbf{D})$ be a conventional AE scheme, and let $\text{KD} : \mathcal{K} \times \{0, 1\}^* \rightarrow \mathcal{K}$ be a key-derivation function of the same key space. For any adversary $\mathcal{A}$ attacking **SE1**[KD, Π], we can construct adversaries $\mathcal{B}$ and $\mathcal{B}'$ of about the same efficiency as $\mathcal{A}$ such that*

$$\text{Adv}_{\text{SE1}[\text{KD}, \Pi]}^{\text{noae2}}(\mathcal{A}) \leq \text{Adv}_{\Pi}^{\text{ae}}(\mathcal{B}) + \text{Adv}_{\text{KD}}^{\text{prf}}(\mathcal{B}') .$$

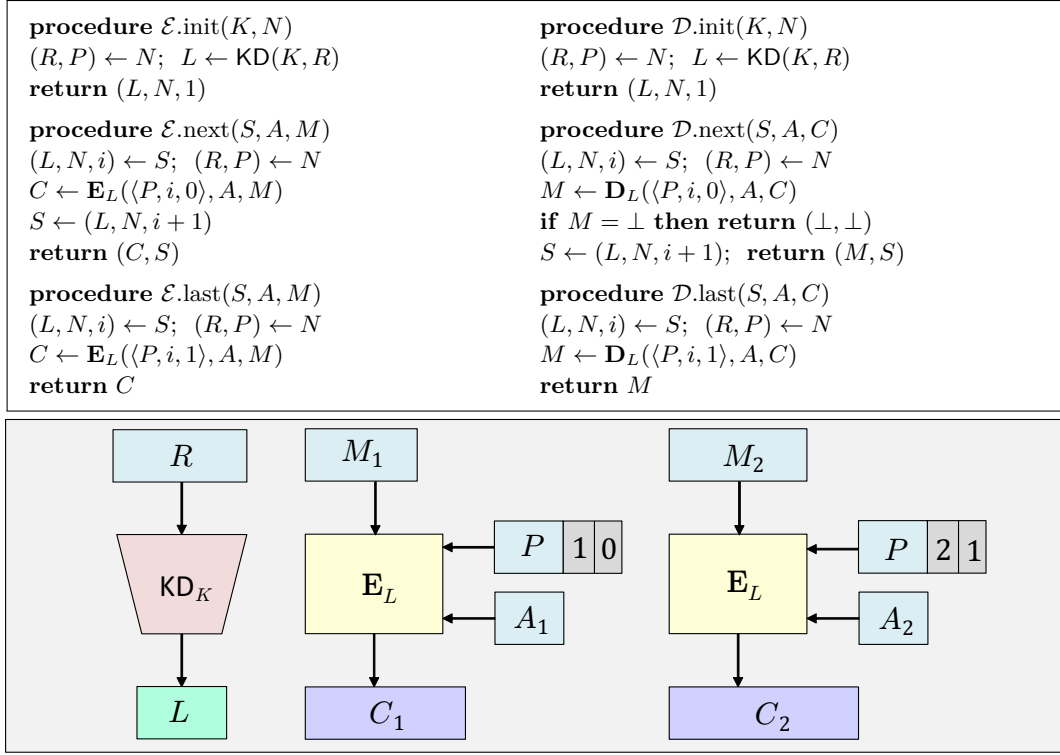


Fig. 8: **The canonical online AE scheme $\text{SE1}[\text{KD}, \Pi] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, where $\Pi = (\mathbf{K}, \mathbf{E}, \mathbf{D})$ is a conventional AE scheme of the same key-generation algorithm, and $\text{KD} : \mathcal{K} \times \{0, 1\}^* \rightarrow \mathcal{K}$ is a key-derivation function.**

Adversary $\mathcal{B}$ makes the same number of encryption queries as $\mathcal{A}$, and at most as many decryption queries as the total number of ciphertext segments in $\mathcal{A}$'s decryption queries. Adversary $\mathcal{B}'$ makes the same number of queries as the number of (partial) streams that $\mathcal{A}$ encrypts or decrypts.

EFFECT OF NONCE RANDOMIZATION. Suppose that in $\text{SE1}[\text{KD}, \Pi]$, we instantiate Π via GCM. In modeling the nOAE2 notion, we assume that nonces are adversarially generated. However, in practice, nonces in encryption queries are generated by legitimate users, although decryption queries might be generated by an adversary. In that case, we can analyze the term $\text{Adv}_{\text{GCM}}^{\text{ae}}(\mathcal{B})$ further based on the implementation choice of the nonces. Assume that nonces for encryption queries of each user are still distinct, which happens with probability at least $1 - q^2/2^{185}$, where q is the number of $\mathcal{A}$'s streams.

For the constructed adversary $\mathcal{B}$ in Theorem 1, if among the nonces $N = (R, P)$ of the encryption queries of the given adversary $\mathcal{A}$, each component P is used for at most d users then $\mathcal{B}$ is d -repeating. (Here $\mathcal{A}$ can still repeat P as many times as it likes for each of the d users.) Additionally, if among the nonces $N = (R, P)$ of the encryption queries for the same user of $\mathcal{A}$, each value R repeats at most c times, then $\mathcal{B}$ encrypts at most $c\ell$ bits per user, where ℓ is maximum length of each (partial) stream that $\mathcal{A}$ encrypts.

Suppose that GCM is built on top of a blockcipher $E : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ that we model as an ideal cipher. From a result of Hoang, Tessaro, and Thiruvengadam [18, Theorem 3.1], $\text{Adv}_{\text{GCM}[E]}^{\text{ae}}(\mathcal{B})$

is in the order of $\frac{(d+n)(p+q)+n\sigma}{2^k} + \frac{\sigma c \ell}{2^n}$, where (i) p is the number of ideal-cipher queries of $\mathcal{B}$, (ii) q is the number of ENC and DEC queries of $\mathcal{B}$, which is also the number of queries of $\mathcal{A}$, and (iii) σ is the total length of the ENC/DEC queries of $\mathcal{B}$, which is also the total length of the (partial) streams that $\mathcal{A}$ encrypts and decrypts.

For Tink’s streaming encryption, recall that a nonce $N = (R, P)$ is generated as follows: (i) the 128-bit prefix of R is picked uniformly at random, and (ii) the string P is a 56-bit uniformly random string. To bound the parameter c , we will use the following balls-into-bins result of Bose, Hoang, and Tessaro [7].

Lemma 1. [7, Lemma 10] *Fix $0 < \epsilon < 1$. Suppose that we throw q balls uniformly at random to 2^m bins, with $q \leq 2^{(1-\epsilon)m-1}$. Then the chance that there is some bin of at least $\lceil 1.5/\epsilon \rceil$ balls is at most $2^{-m/2}$.*

If $\mathcal{A}$ makes at most $q \leq 2^{95}$ queries, using Lemma 1 with $m = 128$ and $\epsilon = 1/4$, with probability at least $1 - 2^{-64}$, we have

$$c \leq \lceil 1.5/\epsilon \rceil - 1 = 5 .$$

Note that randomization is not the only way to make c small. For example, if one implements R as a counter then $c = 1$. To bound the parameter d , we will use the following balls-into-bins result of Hoang and Tessaro [17].

Lemma 2. [17, Lemma 15] *Suppose that we throw q balls uniformly at random to 2^m bins. With probability at most 2^{-m} , there is some bin of at least $\max\{4m, 4q/2^m\}$ balls.*

Using Lemma 2 with $m = 56$, with probability at least $1 - 2^{-56}$, we have $d \leq \max\{224, 4q/2^{56}\}$. Thus Tink’s implementation choice of nonce generation yields a very good bound for $\text{Adv}_{\text{GCM}[E]}^{\text{ae}}(\mathcal{B})$.

DISCUSSION. While incorporating a header H into a nonce may provide some additional hedge against randomness failure, it is unclear how much it helps. On the other hand, it is much faster to authenticate H via an AD segment. Specifically, in Tink’s streaming encryption, H is digested via HMAC-SHA256, whereas under our proposed alternative, H will be processed by GHASH, the underlying universal hash function of GCM. In most platforms, GHASH is several times faster than HMAC-SHA256.

On the other hand, the bound in Theorem 1 contains the term $\text{Adv}_{\text{KD}}^{\text{prf}}(\mathcal{B}')$. If the key length is short then the latter term only admits an adequate bound at best, due to a key-collision attack as mentioned in Section 2. Using random nonces does not improve the PRF bound, because KD is also used for adversarial nonces in decryption queries. This issue seems to be an artifact of a standard-model analysis instead of a real attack. Nevertheless, it can be easily fixed if one uses a 256-bit key for KD; the derived subkeys can still be of 128-bit length if one wants to use AES-128 for GCM.

Next, recall that in Tink’s streaming encryption, the conventional AE scheme Π is GCM, and the latter is built on top of AES. It’s desirable to find an AES-based instantiation of KD (for both AES-128 and AES-256) to reduce the code complexity, and improve the speed of KD. In Section 5, we show how to build such a construction via the XOR construction of Bellare, Krovetz, and Rogaway [2].

Finally, from the analysis above, the **SE1** construction is not optimally designed to resist randomness failure. Security degrades if, for example, R is uniformly random, but P is a constant string,

<pre> procedure INITIALIZE//Games G_1, G_2 $K_1, K_2, \dots \leftarrow \mathcal{K}$; $J_1, J_2, \dots \leftarrow 0$ $f_1, f_2, \dots \leftarrow \text{Func}(\{0, 1\}^*, \mathcal{K})$ procedure Enc.init(i, N) $(R, P) \leftarrow N$; $J_i \leftarrow J_i + 1$; $j \leftarrow J_i$ $L \leftarrow \text{KD}(K_i, R)$ //For game G_1 only $L \leftarrow f_i(R)$ //For game G_2 only $S_{i,j} \leftarrow (L, N, 1)$; return J_i procedure Enc.next(i, j, A, M) if $S_{i,j} = \perp$ then return $\perp$ $(L, N, r) \leftarrow S_{i,j}$; $(R, P) \leftarrow N$ $C \leftarrow \mathbf{E}_L(\langle P, r, 0 \rangle, A, M)$ $S_{i,j} \leftarrow (L, N, r + 1)$; return C procedure Enc.last(i, j, A, M) if $S_{i,j} = \perp$ then return $\perp$ $(L, N, r) \leftarrow S_{i,j}$; $(R, P) \leftarrow N$ $C \leftarrow \mathbf{E}_L(\langle P, r, 1 \rangle, A, M)$; $S_{i,j} \leftarrow \perp$ return C </pre>	<pre> procedure Dec($i, N, A, C, \mathcal{I}, a$) if $A \neq C$ then return false $(R, P) \leftarrow N$; $m \leftarrow C$ $L \leftarrow \text{KD}(K_i, R)$ //For game G_1 only $L \leftarrow f_i(R)$ //For game G_2 only for $r \leftarrow 1$ to $\mathcal{I}$ do $j \leftarrow \mathcal{I}[r]$; if $j > m$ or $j < 1$ then return false if $(a = 0$ or $j < m)$ then if $\mathbf{D}_L(\langle P, j, 0 \rangle, A[j], C[j]) = \perp$ then return false elseif $\mathbf{D}_L(\langle P, m, 1 \rangle, A[m], C[m]) = \perp$ then return false return true procedure FINALIZE(b') return $(b' = 1)$ </pre>
---	---

Fig. 9: Games G_1 and G_2 in the proof of Theorem 1. Each game maintains an internal procedure LOOK.

since d now can be as big as q . In Section 6, we give better canonical online AE designs that are robust against such randomness failure.

4.3 Proof of Theorem 1

Consider games G_1 – G_4 in Fig. 9 and Fig. 10. Game G_1 corresponds to game $\mathbf{G}_{\text{SE1}[\text{KD}, \Pi]}^{\text{noae2}}(\mathcal{A})$ with challenge bit $b = 1$. In game G_1 , each user i has a corresponding key K_i , but in game G_2 , we sample a uniformly random function $f_i : \{0, 1\}^* \rightarrow \mathcal{K}$, and for each call to $\text{KD}(K_i, \cdot)$, we make a corresponding call to $f_i(\cdot)$ instead.

Game G_3 is similar to game G_2 , but in each encryption query for user i and nonce $N = (R, P)$ that results in a call $C \leftarrow \mathbf{E}_L(N^*, A, M)$, we store the tuple (i, R, N^*, A, C) in a table. On the other hand, each time we are supposed to check if $\mathbf{D}_L(N^*, A, C) \neq \perp$ for a user i under nonce $N = (R, P)$, if there is already a prior tuple (i, R, N^*, A, C) then we simply return true. Otherwise we will indeed call $\mathbf{D}_L(N^*, A, C)$.

Game G_4 is similar to game G_3 , but for each call $C \leftarrow \mathbf{E}_L(N^*, A, M)$, we instead pick a truly random C of the same length. Moreover, for each call $\mathbf{D}_L(N^*, A, C)$, we instead use $\perp$ as the answer. Thanks to the restriction on the behavior of the adversary, any call to the Dec oracle must eventually end up with a call to $\mathbf{D}$ instead of looking up prior tuples from encryption queries. As a result, any Dec query will result in a false-answer. Hence game G_4 coincides with game $\mathbf{G}_{\text{SE1}[\text{KD}, \Pi]}^{\text{noae2}}(\mathcal{A})$ of challenge bit $b = 0$, and thus

$$\text{Adv}_{\text{SE1}[\text{KD}, \Pi]}^{\text{noae2}}(\mathcal{A}) = \Pr[G_1(\mathcal{A})] - \Pr[G_4(\mathcal{A})] .$$

We now bound the gap between each pair of consecutive games. Consider the following adversary $\mathcal{B}'$ attacking the (muti-user) PRF security of KD. It runs $\mathcal{A}$ and simulates game G_1 , but for each

<pre> procedure INITIALIZE//Games G_3, G_4 $J_1, J_2, \dots \leftarrow 0$ $f_1, f_2, \dots \leftarrow \text{sFunc}(\{0, 1\}^*, \mathcal{K})$ procedure Enc.init(i, N) $(R, P) \leftarrow N$; $J_i \leftarrow J_i + 1$ $j \leftarrow J_i$; $L \leftarrow f_i(R)$ $S_{i,j} \leftarrow (L, N, 1)$; return J_i procedure Enc.next(i, j, A, M) if $S_{i,j} = \perp$ then return $\perp$ $(L, N, r) \leftarrow S_{i,j}$; $(R, P) \leftarrow N$ $N^* \leftarrow \langle P, r, 0 \rangle$; $C \leftarrow \mathbf{E}_L(N^*, A, M)$ $C \leftarrow \text{s} \{0, 1\}^{ C }$ //For game G_4 only $\text{Tab}[i, R, N^*, A, C] \leftarrow \text{true}$ $S_{i,j} \leftarrow (L, N, r + 1)$; return C procedure Enc.last(i, j, A, M) if $S_{i,j} = \perp$ then return $\perp$ $(L, N, r) \leftarrow S_{i,j}$; $(R, P) \leftarrow N$ $N^* \leftarrow \langle P, r, 1 \rangle$; $C \leftarrow \mathbf{E}_L(N^*, A, M)$ $C \leftarrow \text{s} \{0, 1\}^{ C }$ //For game G_4 only $\text{Tab}[i, R, N^*, A, C] \leftarrow \text{true}$; $S_{i,j} \leftarrow \perp$ return C </pre>	<pre> procedure Dec($i, N, A, C, \mathcal{I}, a$) if $A \neq C$ then return false $(R, P) \leftarrow N$; $m \leftarrow C$ for $r \leftarrow 1$ to $\mathcal{I}$ do $j \leftarrow \mathcal{I}[r]$; if $j > m$ or $j < 1$ then return false if $(a = 0$ or $j < m)$ then if LOOK($i, R, \langle P, j, 0 \rangle, A[j], C[j]$) = false then return false elseif LOOK($i, R, \langle P, m, 1 \rangle, A[m], C[m]$) = false then return false return true private procedure LOOK(i, R, N^*, A, C) if Tab[i, R, N^*, A, C] $\neq \perp$ then return true $L \leftarrow f_i(R)$; $M \leftarrow \mathbf{D}_L(N^*, A, C)$ $M \leftarrow \perp$ //For game G_4 only return ($M \neq \perp$) procedure FINALIZE(b') return ($b' = 1$) </pre>
---	--

Fig. 10: **Games G_3 and G_4 in the proof of Theorem 1.** Each game maintains an internal procedure LOOK.

call to $\text{KD}(K_i, \cdot)$, it instead uses its oracle $\text{EVAL}(i, \cdot)$. Thus

$$\Pr[G_1(\mathcal{A})] - \Pr[G_2(\mathcal{A})] = \text{Adv}_{\text{KD}}^{\text{prf}}(\mathcal{B}') .$$

Next, in game G_3 , for each table look-up $\text{LOOK}(i, R, N^*, A, C)$ with $N^* = \langle P, j, a \rangle$, the triple (i, R, N^*) uniquely determines the user i , nonce $N = (R, P)$, and counter j . Since $\mathcal{A}$ is nonce-respecting, the table look-ups do not return false positives. Thanks to the correctness of the conventional AE scheme Π ,

$$\Pr[G_2(\mathcal{A})] = \Pr[G_3(\mathcal{A})] .$$

Finally, consider the following adversary $\mathcal{B}$ attacking Π . It runs $\mathcal{A}$ and simulates game G_3 , but with the following differences. First, instead of eagerly sampling a function f_i for each user i , adversary $\mathcal{B}$ lazily implements f_i . That is, each time it has to evaluate $f_i(R)$, if this is the first time it has to do this evaluation, it picks $L \leftarrow \text{s} \mathcal{K}$, stores the pair $(\langle i, R \rangle, L)$, and returns L as the answer, where $\langle i, R \rangle$ is a unique encoding of (i, R) as an integer. Otherwise, if there is already some pair $(\langle i, R \rangle, L)$ then it simply returns L as the answer. Next, for each call $\mathbf{E}_L(N^*, A, M)$ for a user i under nonce $N = (R, P)$, it instead calls $\text{ENC}(\langle i, R \rangle, N^*, A, M)$. Likewise, for each call $\mathbf{D}_L(N^*, A, C)$ for a user i under nonce $N = (R, P)$, it calls $\text{DEC}(\langle i, R \rangle, N^*, A, C)$.

Thanks to the table look-ups in game G_3 , adversary $\mathcal{B}$ does not make prohibited queries to the decryption oracle. Moreover, for each encryption query $\text{ENC}(\langle i, R \rangle, N^*, A, M)$ with $N^* = \langle P, j, a \rangle$, the pair $(\langle i, R \rangle, N^*)$ uniquely determines the user i , nonce $N = (R, P)$, and counter j . Since $\mathcal{A}$ is nonce-respecting, $\mathcal{B}$ is nonce-respecting. Moreover,

$$\Pr[G_3(\mathcal{A})] - \Pr[G_4(\mathcal{A})] = \text{Adv}_{\Pi}^{\text{ae}}(\mathcal{B}) .$$

Summing up

$$\begin{aligned} \text{Adv}_{\text{SE1}[\text{KD}, \Pi]}^{\text{noae}2}(\mathcal{A}) &= \Pr[G_1(\mathcal{A})] - \Pr[G_4(\mathcal{A})] \\ &= \sum_{i=1}^3 \Pr[G_i(\mathcal{A})] - \Pr[G_{i+1}(\mathcal{A})] = \text{Adv}_{\text{KD}}^{\text{prf}}(\mathcal{B}') + \text{Adv}_{\Pi}^{\text{ae}}(\mathcal{B}) . \end{aligned}$$

5 A Better Key-Derivation Function

In this section we show a better construction for the key-derivation **KD** of **SE1**. Our construction is based on a blockcipher $E : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ with $k \in \{n, 2n\}$. In practice one would instantiate E via AES, and thus $n = 128$. Moreover, **KD** only needs to digest inputs of a fixed length m , as the header will be processed as an AD segment. For our construction, we require that $m \leq n - 2$; in practice one can let $m = 120$. This is a bit shorter than the current salt length (128 bits) in Tink, but from the analysis in Section 4.2, this does not affect the security bound of **SE1**.

KEY LENGTH EXTENSION. From the discussion in Section 4.2, we want **KD** to take a $2n$ -bit key to avoid key-collision attacks. As a result, our construction of **KD** requires a blockcipher of $2n$ -bit keys. If $k = 2n$ then we can directly use E . Mouha and Luykx [26] show that if we model $E : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ as an ideal cipher then for any adversary $\mathcal{A}$ making at most p ideal-cipher queries and q evaluation queries, $\text{Adv}_E^{\text{prp}}(\mathcal{A}) \leq \frac{q^2 + 2pq}{2^{k+1}}$.

If $k = n$ then the key length of E is too short for our purpose. In that case, we construct from E a blockcipher $G : \{0, 1\}^{k+n} \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ via a variant of the classic DESX paradigm of Rivest. In particular, on a key $K = J \parallel L$, with $|J| = k$ and $|L| = n$, let

$$G_K(X) = E_J(X \oplus L) \oplus L .$$

The design of DESX uses different keys in pre-whitening and post-whitening E , whereas the variant above, which we call DESX1, uses the same key. It is now folklore knowledge that DESX1 offers $(k + n)/2$ bits of multi-user PRP security, but to the best of our knowledge, there is no explicit proof for this claim. For completeness, in Lemma 3 below, we give a proof in the ideal-cipher model. The proof, which is in Appendix C, is a canonical application of the H-coefficient technique [8, 27].

Lemma 3. *Let $E : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a blockcipher that we model as an ideal cipher, and let G be DESX1[E]. Then for any adversary $\mathcal{A}$ that makes at most p ideal-cipher queries and q evaluation queries,*

$$\text{Adv}_G^{\text{prp}}(\mathcal{A}) \leq \frac{2pq + q^2}{2^{k+n}} .$$

THE XOR CONSTRUCTION. Our main building block is the XOR-of-permutation construction of Bellare, Krovetz, and Rogaway [2]. Let $G : \mathcal{K} \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a blockcipher. Then $\text{XOR}[G] : \mathcal{K} \times \{0, 1\}^{n-1} \rightarrow \{0, 1\}^n$ is constructed via

$$\text{XOR}[G](K, X) = G_K(X \parallel 0) \oplus G_K(X \parallel 1) .$$

If G is DESX1[E] then under the XOR construction, the two post-whitenings will cancel each other out. Thus one can drop the post-whitening of E in this setting, saving two xor's. Theorem 2 below shows that the XOR construction has excellent PRF security; the proof is in Appendix D.

Theorem 2. *Let $n \geq 8$ be an integer. Let $G : \mathcal{K} \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a blockcipher. Then for any adversary $\mathcal{A}$ that makes $q \leq 2^n/128$ evaluation queries, we can construct an adversary $\mathcal{B}$ of about the same efficiency as $\mathcal{A}$ and at most $2q$ queries such that*

$$\text{Adv}_{\text{XOR}[G]}^{\text{prf}}(\mathcal{A}) \leq \text{Adv}_G^{\text{prp}}(\mathcal{B}) + \frac{8\sqrt{n} \cdot q}{2^n} .$$

BUILDING KD. Assume that we have a good PRF construction $F : \mathcal{K} \times \{0, 1\}^{n-1} \rightarrow \{0, 1\}^n$ which we can achieve via the XOR construction. We now show how to build $\text{KD} : \mathcal{K} \times \{0, 1\}^m \rightarrow \{0, 1\}^k$, with $k \in \{n, 2n\}$. For a string X with $|X| < n - 1$, let $X \parallel 0^*$ denote $X \parallel 0^{n-1-|X|}$. If $k = n$ then

$$\text{KD}(K, X) = F_K(X \parallel 0^*) ,$$

and if $k = 2n$ then

$$\text{KD}(K, X) = F_K(X \parallel 0 \parallel 0^*) \parallel F_K(X \parallel 1 \parallel 0^*) .$$

From the fact that F is a good PRF and thanks to the use of the domain separation, KD is also a good PRF.

6 Better Online AE Schemes

In this section, we give two alternative canonical online AE schemes that are better than **SE1**.

6.1 Scheme SE2

THE SCHEME. Let $\Pi = (\mathbf{K}, \mathbf{E}, \mathbf{D})$ be a conventional AE scheme of k -bit key length. We associate to it an encoding mechanism $\langle \cdot \rangle$ that turns a triple (P, i, a) of a nonce prefix P , a counter i , and a flag a into a nonce of Π . Assume that under this encoding, a nonce prefix must have bit length ℓ . For example, if we instantiate Π via GCM and use four bytes to encode a counter and one byte for the flag as in Tink, then $\ell = 56$. Let $\text{KD} : \mathcal{K} \times \mathcal{N} \rightarrow \{0, 1\}^k \times \{0, 1\}^\ell$ be a key-derivation function. The scheme $\text{SE2}[\text{KD}, \Pi] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ of the same key space as KD is specified in Fig. 11; it uses the message space $\mathcal{N}$ of KD as its nonce space.

Informally, **SE2** is similar to **SE1**, but we use KD to digest the entire nonce. Moreover, each nonce prefix P is a part of KD's output, instead of a part of the nonce N .

ANALYSIS OF SE2. The following result shows that $\text{SE2}[\text{KD}, \Pi]$ meets the nOAE2 notion, provided that KD is a good PRF and Π is a good conventional AE scheme. The proof, which is in Appendix E, is similar to the proof of Theorem 1.

Theorem 3. *Let $\Pi = (\mathbf{K}, \mathbf{E}, \mathbf{D})$ be a conventional AE scheme, and let $\text{KD} : \mathcal{K} \times \mathcal{N} \rightarrow \{0, 1\}^k \times \{0, 1\}^\ell$ be a key-derivation function. For any adversary $\mathcal{A}$ attacking $\text{SE2}[\text{KD}, \Pi]$ that makes at most q queries, we can construct adversaries $\mathcal{B}$ and $\mathcal{B}'$ of about the same efficiency as $\mathcal{A}$ such that*

$$\text{Adv}_{\text{SE2}[\text{KD}, \Pi]}^{\text{noae2}}(\mathcal{A}) \leq \text{Adv}_\Pi^{\text{ae}}(\mathcal{B}) + \text{Adv}_{\text{KD}}^{\text{prf}}(\mathcal{B}') .$$

Adversary $\mathcal{B}$ makes the same number of encryption queries as $\mathcal{A}$, and at most as many decryption queries as the total number of ciphertext segments in $\mathcal{A}$'s decryption queries. Adversary $\mathcal{B}'$ makes the same number of queries as the number of (partial) streams that $\mathcal{A}$ encrypts or decrypts. Moreover, with probability at least $1 - 2^{-\ell}$, adversary $\mathcal{B}$ is $\max\{4\ell, 4q/2^\ell\}$ -repeating.

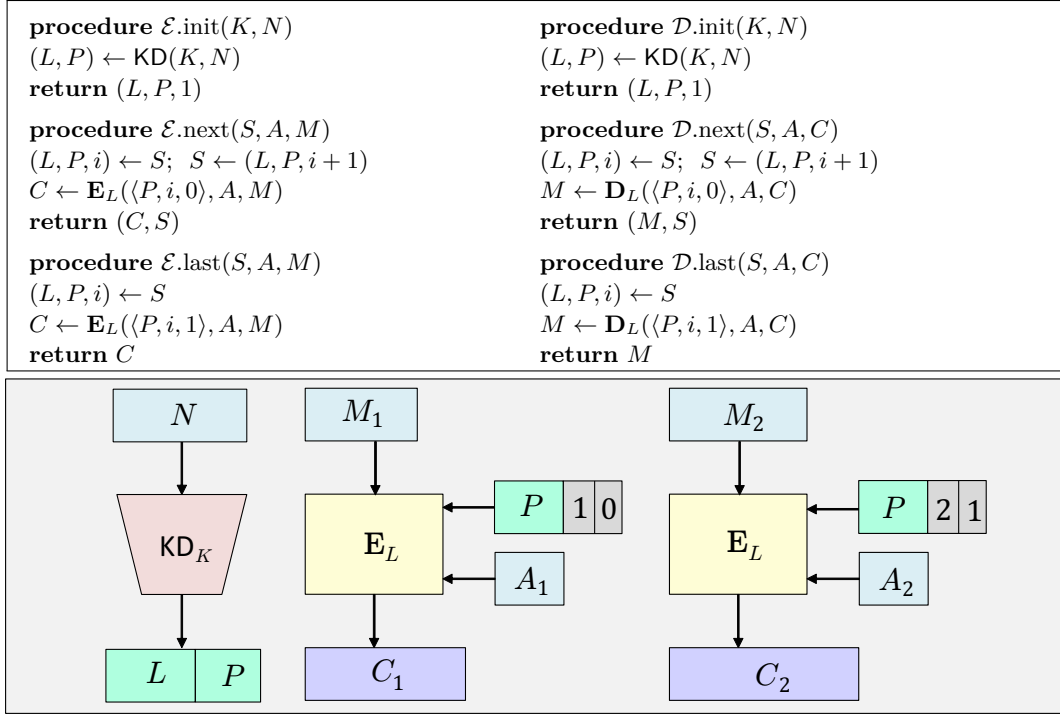


Fig. 11: The canonical online AE scheme $\mathbf{SE2}[\text{KD}, \Pi] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ that is built on top on a conventional AE scheme $\Pi = (\mathbf{K}, \mathbf{E}, \mathbf{D})$ and a key-derivation function KD .

DISCUSSION. Note that in Theorem 3, the constructed adversary $\mathcal{B}$ is $\max\{4\ell, 4q/2^\ell\}$ -repeating, provided that nonces for each user are unique. This makes $\mathbf{SE2}$ more robust to randomness failure than $\mathbf{SE1}$. However, this claim is under an implicit assumption (lying beneath the term $\text{Adv}_{\text{KD}}^{\text{prf}}(\mathcal{B}')$) that the master key is 256-bit long, Note that derived subkeys L can still be of 128-bit if one wants to use AES-128 for GCM.

As Tink aims to support both AES-GCM-128 and AES-GCM-256, KD may have to produce 39 bytes. The simplest way is to instantiate KD via HMAC-SHA384. (If one wants to instantiate KD via HMAC-SHA256 then it may require two hash calls.)

For applications that can tolerate 15-byte nonces then one can instantiate KD via the XOR construction in Section 5. Specifically, suppose that we are given a PRF $F : \mathcal{K} \times \{0, 1\}^{127} \rightarrow \{0, 1\}^{128}$ which we can directly instantiate via the XOR construction. For a string X with $|X| < 127$, let $X \parallel 0^*$ denote $X \parallel 0^{127-|X|}$. If $k = 128$ and $\ell = 56$ then

$$\text{KD}(K, X) \leftarrow F_K(X \parallel 0 \parallel 0^*) \parallel F_K(X \parallel 1 \parallel 0^*)[1 : 56] . \quad (1)$$

If $k = 256$ and $\ell = 56$ then we let $\text{KD}(K, X)$ to be

$$F_K(X \parallel 00 \parallel 0^*) \parallel F_K(X \parallel 01 \parallel 0^*) \parallel F_K(X \parallel 10 \parallel 0^*)[1 : 56] . \quad (2)$$

Hence we would need 2 or 3 calls to the XOR construction, which amounts to 4 or 6 (fully parallelizable) AES calls. Thus KD has the same speed as the key-derivation of AES-GCM-SIV [13].

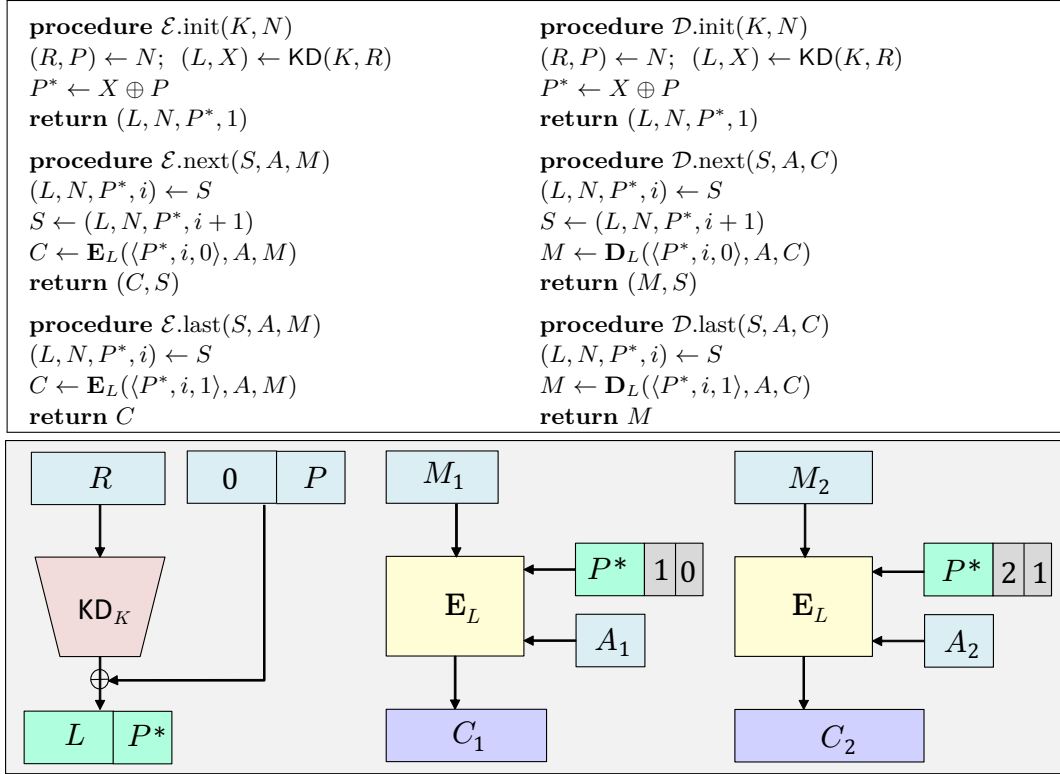


Fig. 12: The canonical online AE scheme $\mathbf{SE3}[\text{KD}, \Pi] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ that is built on top of a conventional AE scheme $\Pi = (\mathbf{K}, \mathbf{E}, \mathbf{D})$ and a key-derivation function KD .

For applications that need longer nonces, it is unclear how to build a beyond-birthday-bound KD efficiently via AES; we leave it as an open problem. In Section 6.2, we give an alternative canonical online AE scheme $\mathbf{SE3}$ that has the best of both worlds: (i) its nOAE2 security is as good as $\mathbf{SE2}$, and (ii) it supports long nonces, say 22 bytes, and (iii) its key-derivation can be built as indicated in Eq. (1) and Eq. (2).

6.2 Scheme $\mathbf{SE3}$

THE SCHEME. Let $\Pi = (\mathbf{K}, \mathbf{E}, \mathbf{D})$ be a conventional AE scheme of k -bit key length. We associate to it an encoding mechanism $\langle \cdot \rangle$ that turns a triple (P, i, a) of a nonce prefix P , a counter i , and a flag a into a nonce of Π . Assume that under this encoding, a nonce prefix must have bit length ℓ . For example, if we instantiate Π via GCM and use four bytes to encode a counter and one byte for the flag as in Tink, then $\ell = 56$. Let $\text{KD} : \mathcal{K} \times \{0, 1\}^s \rightarrow \{0, 1\}^k \times \{0, 1\}^\ell$ be a key-derivation function. The scheme $\mathbf{SE3}[\text{KD}, \Pi] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ of the same key space as KD is specified in Fig. 12; its nonce space is $\{0, 1\}^s \times \{0, 1\}^\ell$.

Informally, a nonce N in $\mathbf{SE3}$ is still a pair (R, P) and we feed only R to KD , as in $\mathbf{SE1}$. However, in addition to a subkey L , we also derive a secret mask X , and use that to whiten the nonce prefix P , obtaining an effective nonce prefix $P^* \leftarrow P \oplus X$. Nonces for the conventional AE scheme Π are based on P^* instead.

In instantiating KD, one can let $s = 120$ and $\ell = 56$, meaning that nonces of **SE3** are 22-byte long. One then can instantiate KD as indicated in Eq. (1) and Eq. (2). Again, we recommend using a 256-bit key for KD to resist key-collision attacks.

ANALYSIS OF SE3. Theorem 4 below confirms that **SE3** is as good as **SE2**. Below, we sketch some proof ideas; the full proof is in Appendix E.

Theorem 4. *Let $\Pi = (\mathbf{K}, \mathbf{E}, \mathbf{D})$ be a conventional AE scheme, and let $\text{KD} : \mathcal{K} \times \{0, 1\}^s \rightarrow \{0, 1\}^k \times \{0, 1\}^\ell$ be a key-derivation function. For any adversary $\mathcal{A}$ attacking **SE3**[KD, Π] that makes at most q queries, we can construct adversaries $\mathcal{B}$ and $\mathcal{B}'$ of about the same efficiency as $\mathcal{A}$ such that*

$$\text{Adv}_{\text{SE3}[\text{KD}, \Pi]}^{\text{noae2}}(\mathcal{A}) \leq \text{Adv}_{\Pi}^{\text{ae}}(\mathcal{B}) + \text{Adv}_{\text{KD}}^{\text{prf}}(\mathcal{B}') + 2^{-\ell}.$$

Adversary $\mathcal{B}$ makes the same number of encryption queries as $\mathcal{A}$, and at most as many decryption queries as the total number of ciphertext segments in $\mathcal{A}$'s decryption queries. Adversary $\mathcal{B}'$ makes the same number of queries as the number of (partial) streams that $\mathcal{A}$ encrypts or decrypts. Moreover, adversary $\mathcal{B}$ is $\max\{4\ell, 4q/2^\ell\}$ -repeating.

PROOF IDEAS. It is tricky to prove that the constructed adversary $\mathcal{B}$ is d -repeating, where $d = \max\{4\ell, 4q/2^\ell\}$.

First, one can only use balls-into-bins analysis here if $\mathcal{A}$ *non-adaptively* picks its queries. If the adversary somehow can adaptively learn the effective nonce prefixes via its queries then it can make $\mathcal{B}$'s nonces repeat among as many users as it likes. To deal with this issue, we employ a trick in [18] for constructing $\mathcal{B}$. Specifically, $\mathcal{B}$ keeps track of the max load of its bin, and terminates prematurely with output 1 when some bin contains d balls. In the real world, $\mathcal{B}$ is supposed to output 1, and thus the premature termination only increases the chance that its guess is correct. In the ideal world of $\mathcal{B}$, whatever $\mathcal{A}$ receives are independent of the effective nonce prefixes, and thus its queries are non-adaptive. We will then use a balls-into-bins argument to bound the probability of premature termination in the ideal world.

Next, we cannot immediately use Lemma 2 because in our situation, the throws are *dependent*. For example, for two encryption queries of the same user with nonces $N_1 = (R, P_1)$ and $N_2 = (R, P_2)$ such that $P_1 \neq P_2$, their effective nonce prefixes P_1^* and P_2^* must be different, meaning that the corresponding two balls must go to different bins. To resolve this issue, we observe that in this setting, for any d balls, the chance that they all go to the same bin is at most $2^{(1-d)\ell}$. This is actually the only hypothesis that the proof of Lemma 2 in [17] needs, although its statement is given for uniformly thrown balls.

7 Conclusions

We give a multi-user analysis of the streaming encryption in Google's Tink library, by viewing it as a canonical online AE scheme **SE1**. We show that **SE1** is secure for any proper way to generate nonces, but Tink's choice of random nonces, coupled with a nonce-based key-derivation function KD, indeed improves the concrete security bound. Motivated by the fact that **SE1**'s design is not robust against randomness failure, we propose better designs **SE2** and **SE3**. In addition, instead of relying on HMAC-SHA256, we show how to efficiently instantiate KD from AES via the XOR construction. Our instantiation of KD is as fast as the key-derivation function of AES-GCM-SIV.

Based on our analysis, we recommend that Tink’s streaming encryption follows **SE3** instead of **SE1** to resist randomness failure. Moreover, if one has a header to authenticate, one should treat it as a segmented AD, instead of feeding it to the key-derivation function KD. This also allows one to instantiate KD via AES, and thus improve both code complexity and efficiency. Finally, one should use a 256-bit master key to avoid key-collision attacks, but internally, one can still use AES-128 for both KD (via the DESX1 paradigm) and GCM (by deriving a 128-bit subkey).

Acknowledgement

We thank CCS reviewers for useful feedback, especially for the information that a major application of Tink’s streaming encryption is to encrypt large files, which requires random-access decryption. Viet Tung Hoang was supported in part by NSF grants CICI-1738912 and CRII-1755539. Yaobin Shen was supported in part by National Key Research and Development Program of China (No. 2019YFB2101601) and China Scholarship Council (No. 201806230107). Part of this work was done while Yaobin Shen was visiting Florida State University.

References

1. M. Bellare and R. Impagliazzo. A tool for obtaining tighter security analyses of pseudorandom function based constructions, with applications to PRP to PRF conversion. Cryptology ePrint Archive, Report 1999/024, 1999. <http://eprint.iacr.org/1999/024>.
2. M. Bellare, T. Krovetz, and P. Rogaway. Luby-Rackoff backwards: Increasing security by making block ciphers non-invertible. In K. Nyberg, editor, *EUROCRYPT’98*, volume 1403 of *LNCS*, pages 266–280. Springer, Heidelberg, May / June 1998.
3. M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006.
4. M. Bellare and B. Tackmann. The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 247–276. Springer, Heidelberg, Aug. 2016.
5. E. Biham. How to decrypt or even substitute DES-encrypted messages in 2^{28} steps. *Information Processing Letters*, 84(3):117–124, 2002.
6. P. Bose, V. T. Hoang, and S. Tessaro. Revisiting AES-GCM-SIV: Multi-user security, faster key derivation, and better bounds. In *EUROCRYPT 2018*, 2018.
7. P. Bose, V. T. Hoang, and S. Tessaro. Revisiting AES-GCM-SIV: Multi-user security, faster key derivation, and better bounds. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 468–499. Springer, Heidelberg, Apr. / May 2018.
8. S. Chen and J. P. Steinberger. Tight security bounds for key-alternating ciphers. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 327–350. Springer, Heidelberg, May 2014.
9. B. Cogliati. Tweaking a block cipher: multi-user beyond-birthday-bound security in the standard model. *Designs, Codes and Cryptography*, 86(12):2747–2763, 2018.
10. W. Dai, V. T. Hoang, and S. Tessaro. Information-theoretic indistinguishability via the Chi-Squared method. In *CRYPTO 2017*, 2017.
11. L. Dorrendorf, Z. Gutterman, and B. Pinkas. Cryptanalysis of the windows random number generator. In P. Ning, S. De Capitani di Vimercati, and P. F. Syverson, editors, *ACM CCS 2007*, pages 476–485. ACM Press, Oct. 2007.
12. Google. Tink library, 2020. <https://github.com/google/tink>.
13. S. Gueron and Y. Lindell. Better bounds for block cipher modes of operation via nonce-based key derivation. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *ACM CCS 2017*, pages 1019–1036. ACM Press, Oct. / Nov. 2017.

14. N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In T. Kohno, editor, *USENIX Security 2012*, pages 205–220. USENIX Association, Aug. 2012.
15. V. T. Hoang, R. Reyhanitabar, P. Rogaway, and D. Vizár. Online authenticated-encryption and its nonce-reuse misuse-resistance. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 493–517. Springer, Heidelberg, Aug. 2015.
16. V. T. Hoang and S. Tessaro. Key-alternating ciphers and key-length extension: Exact bounds and multi-user security. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 3–32. Springer, Heidelberg, Aug. 2016.
17. V. T. Hoang and S. Tessaro. The multi-user security of double encryption. In J. Coron and J. B. Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 381–411. Springer, Heidelberg, Apr. / May 2017.
18. V. T. Hoang, S. Tessaro, and A. Thiruvengadam. The multi-user security of GCM, revisited: Tight bounds for nonce randomization. In *CCS 2018*, 2018.
19. A. Jha and M. Nandi. Tight security of cascaded lrw2. *Journal of Cryptology*, pages 1–46, 2020.
20. J. Kilian and P. Rogaway. How to protect DES against exhaustive key search. In N. Kobitz, editor, *CRYPTO’96*, volume 1109 of *LNCS*, pages 252–267. Springer, Heidelberg, Aug. 1996.
21. T. Krovetz and P. Rogaway. The software performance of authenticated-encryption modes. In A. Joux, editor, *FSE 2011*, volume 6733 of *LNCS*, pages 306–327. Springer, Heidelberg, Feb. 2011.
22. A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, and C. Wachter. Public keys. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 626–642. Springer, Heidelberg, Aug. 2012.
23. S. Lucks. The sum of PRPs is a secure PRF. In B. Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 470–484. Springer, Heidelberg, May 2000.
24. D. A. McGrew and J. Viega. The security and performance of the Galois/counter mode (GCM) of operation. In A. Canteaut and K. Viswanathan, editors, *INDOCRYPT 2004*, volume 3348 of *LNCS*, pages 343–355. Springer, Heidelberg, Dec. 2004.
25. R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge University Press, 1995.
26. N. Mouha and A. Luykx. Multi-key security: The Even-Mansour construction revisited. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 209–223. Springer, Heidelberg, Aug. 2015.
27. J. Patarin. The “coefficients H” technique (invited talk). In R. M. Avanzi, L. Keliher, and F. Sica, editors, *SAC 2008*, volume 5381 of *LNCS*, pages 328–345. Springer, Heidelberg, Aug. 2009.
28. J. Patarin. Introduction to mirror theory: Analysis of systems of linear equalities and linear non equalities for cryptography. Cryptology ePrint Archive, Report 2010/287, 2010. <http://eprint.iacr.org/2010/287>.
29. P. Rogaway, M. Bellare, J. Black, and T. Krovetz. OCB: A block-cipher mode of operation for efficient authenticated encryption. In M. K. Reiter and P. Samarati, editors, *ACM CCS 2001*, pages 196–205. ACM Press, Nov. 2001.

A nOAE Does Not Imply nOAE2

In this section, we will show a canonical online AE scheme Π^* that is nOAE-secure, but trivially broken under the nOAE2 notion. In particular, Π is a variant of the **STREAM** construction; its procedures $\mathcal{E}.\text{last}$ and $\mathcal{D}.\text{last}$ are modified to preserve the nOAE security, but damage the nOAE2 security. The code is given in Fig. 13. The change here is that in $\mathcal{E}.\text{last}$, in addition to the ciphertext segment $C_0 \leftarrow \mathbf{E}_K(\langle N, i, 1 \rangle, A, M)$, we will release an additional ciphertext segment $C_1 \leftarrow \mathbf{E}_K(\langle N, i + 1, 0 \rangle, A, M)$. Procedure $\mathcal{D}.\text{last}$ will attempt to decrypt both C_0 and C_1 , and if both ciphertexts are valid then it returns the decrypted message of C_0 .

Releasing C_1 does no harm to nOAE security. Indeed, as C_1 can only be decrypted properly under nonce $\langle N, i + 1, 0 \rangle$, the adversary can only use it in a decryption query $\text{Dec}(u, N, \mathbf{A}, \mathbf{C}, a)$ with $\mathbf{C}[i + 1] = C_1$, where u is the user of the corresponding encryption query. But since decryption is in order, the adversary will never pass the decryption of $\mathbf{C}[i]$, as it does not know any valid ciphertext segment under nonce $\langle N, i, 0 \rangle$ for user u .

procedure $\mathcal{E}.\text{init}(K, N)$ return $(K, N, 1)$ procedure $\mathcal{E}.\text{next}(S, A, M)$ $(K, N, i) \leftarrow S; S \leftarrow (K, N, i + 1)$ $C \leftarrow \mathbf{E}_K(\langle N, i, 0 \rangle, A, M)$ return (C, S) procedure $\mathcal{E}.\text{last}(S, A, M)$ $(K, N, i) \leftarrow S$ $C_0 \leftarrow \mathbf{E}_K(\langle N, i, 1 \rangle, A, M)$ $C_1 \leftarrow \mathbf{E}_K(\langle N, i + 1, 0 \rangle, A, M)$ return (C_0, C_1)	procedure $\mathcal{D}.\text{init}(K, N)$ return $(K, N, 1)$ procedure $\mathcal{D}.\text{next}(S, A, C)$ $(K, N, i) \leftarrow S; S \leftarrow (K, N, i + 1)$ $M \leftarrow \mathbf{D}_K(\langle N, i, 0 \rangle, A, C)$ return (M, S) procedure $\mathcal{D}.\text{last}(S, A, C)$ $(K, N, i) \leftarrow S; (C_0, C_1) \leftarrow C$ $M \leftarrow \mathbf{D}_K(\langle N, i + 1, 0 \rangle, A, C_1)$ if $M = \perp$ then return $\perp$ else return $\mathbf{D}_K(\langle N, i, 1 \rangle, A, C_0)$
---	--

Fig. 13: A variant Π^* of the **STREAM** construction that is nOAE-secure, but trivially broken under nOAE2. It is built on top of a conventional AE scheme $\Pi = (\mathbf{K}, \mathbf{E}, \mathbf{D})$.

In contrast, it is trivial to break the nOAE2 security of Π^* . The adversary first calls $\mathcal{E}.\text{init}(1, N)$, for an arbitrary nonce N , to get an index j . It then calls $\mathcal{E}.\text{last}(1, j, A, M)$ for an arbitrary AD A and message M to get a ciphertext segment (C_0, C_1) . Finally, it makes a decryption query $\text{Dec}(1, N, \mathbf{A}, \mathbf{C}, \mathcal{I}, 0)$, with $\mathbf{A} = (A, A)$ and $\mathbf{C} = (C_1, C_1)$, and $\mathcal{I} = (2)$, meaning that it only asks for a decryption of the second ciphertext segment. The adversary then outputs 1 if the decryption oracle returns true, and outputs 0 otherwise. Note that in the real world, the first segment of $\mathbf{C}$ is invalid but it never gets checked, and the second segment of $\mathbf{C}$ is decrypted properly. Hence the adversary wins with advantage 1.

B Key-Collision Attacks

In this section we show how to use Biham's key-collision attack on DES [5] to break the (multi-user) PRF and AE security.

B.1 PRF Attack

Let $F : \{0, 1\}^k \times \mathcal{M} \rightarrow \{0, 1\}^n$ be a keyed function. Let r be an arbitrary integer such that $r \geq \lceil (k + 2)/n \rceil$. Let $p, q \geq 1$ be integers such that $pq \leq 2^k$.

THE ATTACK. Adversary $\mathcal{A}$ first picks arbitrary distinct messages $M_1, \dots, M_r \in \mathcal{M}$ and then queries $C_{i,1} \leftarrow \text{EVAL}(i, M_1), \dots, C_{i,r} \leftarrow \text{EVAL}(i, M_r)$ for every $i = 1, \dots, q$. It then picks arbitrary distinct keys $K_1, \dots, K_p \in \{0, 1\}^k$, and computes $C_{j,1}^* \leftarrow F(K_j, M_1), \dots, C_{j,r}^* \leftarrow F(K_j, M_r)$ for every $j = 1, \dots, p$. If there are $i \leq q$ and $j \leq p$ such that $C_{i,\ell} = C_{j,\ell}^*$ for every $\ell = 1, \dots, r$ then the adversary outputs 1, otherwise the adversary outputs 0.

REAL WORLD'S ANALYSIS. In the real world, if there is an adversarial key K_j that coincides with the key of user i then the adversary will output 1. Since the user keys are chosen uniformly at random, this happens with probability

$$1 - \left(1 - \frac{p}{2^k}\right)^q.$$

To simplify this bound, we need the following technical result from [17].

Lemma 4. [17] Let $q \geq 1$ be an integer and $0 < a \leq 1/q$. Then

$$(1 - a)^q \leq 1 - qa/2 .$$

Using Lemma 4 with $a = p/2^k$ indicates that in the real world, the chance that the adversary outputs 1 is at least $\frac{pq}{2^{k+1}}$.

IDEAL WORLD'S ANALYSIS. Assume that there are no two keys K_i and K_j such that $(C_{i,1}^*, \dots, C_{i,r}^*) = (C_{j,1}^*, \dots, C_{j,r}^*)$. For real PRF constructions, this assumption is very likely to hold for an appropriately large r . In the ideal world, the oracle answers are independent, truly random strings, and thus the chance that the adversary outputs 1 is at most

$$1 - \left(1 - \frac{p}{2^{nr}}\right)^q \leq 1 - \left(1 - \frac{p}{2^{k+2}}\right)^q \leq \frac{pq}{2^{k+2}} ,$$

where the first inequality is due to the fact that $nr \geq k + 2$, and the second inequality is due to Bernoulli's inequality.

WRAPPING UP. Summing up, the adversary's advantage is at least

$$\frac{pq}{2^{k+1}} - \frac{pq}{2^{k+2}} = \frac{pq}{2^{k+2}} .$$

B.2 AE Attack

Let $\Pi = (\mathbf{K}, \mathbf{E}, \mathbf{D})$ be a conventional AE scheme of k -bit key length and s -bit nonce length. Let $p, q \geq 1$ be integers such that $pq \leq 2^k$.

THE ATTACK. Let A be the empty-string AD. The adversary $\mathcal{A}$ first picks an arbitrary message M such that $|M| \geq k + 2$, and an arbitrary nonce N . It then queries $C_i \leftarrow \text{ENC}(i, N, A, M)$ for every $i = 1, \dots, q$. It then picks arbitrary distinct keys $K_1, \dots, K_p \in \{0, 1\}^k$, and computes $C_j^* \leftarrow \mathbf{E}(K_j, N, A, M)$ for every $j = 1, \dots, p$. If there are $i \leq q$ and $j \leq p$ such that $C_i = C_j^*$ then the adversary outputs 1, otherwise the adversary outputs 0.

REAL WORLD'S ANALYSIS. In the real world, if there is an adversarial key K_j that coincides with the key of user i then the adversary will output 1. Since the user keys are chosen uniformly at random, this happens with probability

$$1 - \left(1 - \frac{p}{2^k}\right)^q \geq \frac{pq}{2^{k+1}} .$$

where the last inequality is due to Lemma 4. Hence in the real world, the adversary outputs 1 with probability at least $pq/2^{k+1}$.

IDEAL WORLD'S ANALYSIS. Assume that there are no two keys K_i and K_j such that $C_i^* = C_j^*$. For real AE schemes, this assumption is very likely to hold if M is long enough. In the ideal world, the oracle answers are independent, truly random strings, and thus the chance that the adversary outputs 1 is at most

$$1 - \left(1 - \frac{p}{2^{k+2}}\right)^q \leq \frac{pq}{2^{k+2}} ,$$

procedure INITIALIZE $K_1, K_2, \dots \leftarrow \mathcal{K}; \quad b \leftarrow \{0, 1\}$ $\pi_1, \pi_2, \dots \leftarrow \text{Perm}(n)$ procedure FINALIZE(b') return ($b' = b$)	procedure ENC(i, X) $Y_1 \leftarrow E(K_i, X); \quad Y_0 \leftarrow \pi_i(X)$ return Y_b procedure DEC(i, Y) $X_1 \leftarrow E^{-1}(K_i, Y); \quad X_0 \leftarrow \pi_i^{-1}(Y)$ return X_b
--	--

Fig. 14: Game $\mathbf{G}_E^{\pm\text{prp}}$ defining multi-user prp security of a blockcipher E .

where the inequality is due to Bernoulli's inequality.

WRAPPING UP. Summing up, the adversary's advantage is at least

$$\frac{pq}{2^{k+1}} - \frac{pq}{2^{k+2}} = \frac{pq}{2^{k+2}} \ .$$

C Proof of Lemma 3

Our proof is based on the H-coefficient technique of Patarin [8, 27] that we briefly review below.

THE H-COEFFICIENT TECHNIQUE. The H-coefficient technique considers a deterministic distinguisher $\mathcal{A}$ that tries to distinguish a “real” system $\mathbf{S}_1$ from an “ideal” system $\mathbf{S}_0$. The adversary's interactions with those systems define transcripts $\mathcal{T}_1$ and $\mathcal{T}_0$, respectively. The following Lemma 5 bounds the statistical distance $\text{SD}(\mathcal{T}_1, \mathcal{T}_0)$ between $\mathcal{T}_1$ and $\mathcal{T}_0$, namely the best distinguishing advantage of $\mathcal{A}$ between $\mathbf{S}_1$ and $\mathbf{S}_0$.

Lemma 5. [8, 27] *Suppose we can partition the set of valid transcripts for the ideal system into good and bad ones. Further, suppose that there exists $\epsilon \geq 0$ such that $1 - \frac{\text{ps}_{\mathbf{S}_1}(\tau)}{\text{ps}_{\mathbf{S}_0}(\tau)} \leq \epsilon$ for every good transcript τ . Then,*

$$\text{SD}(\mathcal{T}_1, \mathcal{T}_0) \leq \epsilon + \Pr[\mathcal{T}_0 \text{ is bad}] \ .$$

STRONG MULTI-USER PRP. We will prove a better result that DESX1 is a (multi-user) *strong PRP*. We now define the notion. Let $E : \mathcal{K} \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a blockcipher. Let $\text{Perm}(n)$ be the set of all permutations on $\{0, 1\}^n$. For an adversary $\mathcal{A}$, let

$$\text{Adv}_E^{\pm\text{prp}}(\mathcal{A}) = 2 \Pr[\mathbf{G}_E^{\pm\text{prp}}(\mathcal{A})] - 1 \ ,$$

be the advantage of the adversary against the multi-user strong PRP security of E , where game $\mathbf{G}_E^{\pm\text{prp}}$ is defined in Fig. 14. This game is written as a standard-model notion; in the ideal-cipher model, the adversary will be given oracle access to E and its inverse as well. The (ordinary) PRP notion is simply a special case of the strong PRP notion where adversaries are not allowed to make DEC queries.

RESULTS. The following Proposition 2 shows that DESX1 is a good strong PRP, which implies Lemma 3.

Proposition 2. Let $E : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a blockcipher that we model as an ideal cipher, and let $G \leftarrow \text{DESX1}[E]$. Then for any adversary $\mathcal{A}$ that makes at most p ideal-cipher queries and q ENC/DEC queries,

$$\text{Adv}_G^{\pm\text{prp}}(\mathcal{A}) \leq \frac{2pq + q^2}{2^{k+n}} .$$

Proof. Our proof is based on the H-coefficient technique. We will consider even a computationally unbounded adversary, and thus we can assume that the adversary is deterministic and never repeats a prior query. Assume further that the adversary never makes a redundant query: (i) if it queries $Y \leftarrow \text{ENC}(i, X)$ then it won't query $\text{DEC}(i, Y)$ and vice versa, and (ii) if it queries $Y \leftarrow E(J, X)$ then it won't query $E^{-1}(J, Y)$ and vice versa. The real system corresponds to game $\mathbf{G}_G^{\pm\text{prp}}$ with challenge bit $b = 1$, and the ideal system corresponds to game $\mathbf{G}_G^{\pm\text{prp}}$ with challenge bit $b = 0$.

SETUP. In the real world, after the adversary finishes querying, we will give it the keys $K_i = (J_i, L_i)$ of all users i that the adversary queries. In the ideal world, we instead give the adversary truly random strings $K_i \leftarrow \{0,1\}^{k+n}$, independent of the transcript. This key revealing only helps the adversary. Thus a transcript consists of the revealed keys and the following information:

- **Ideal-cipher queries:** For each query $Y \leftarrow E(J, X)$ or $X \leftarrow E^{-1}(J, Y)$, we associate it with an entry (prim, J, X, Y) .
- **Evaluation queries:** For each query $Y \leftarrow \text{ENC}(i, X)$ or $X \leftarrow \text{DEC}(i, Y)$, we associate it with an entry (eval, i, X, Y) .

DEFINING BAD TRANSCRIPTS. We say that a transcript is *bad* if one of the following happens

- There are entries (prim, J, X, Y) and $(\text{eval}, i, X^*, Y^*)$ such that (i) $J = J_i$ and (ii) either $X^* = X \oplus L_i$ or $Y^* = Y \oplus L_i$, where $K_i = J_i \parallel L_i$ is the key of user i as indicated by the transcript. In the real world these entries must be consistent, but it is unlikely in the ideal world.
- There are entries (eval, i, X, Y) and $(\text{eval}, j, X^*, Y^*)$, with $i \neq j$, such that (i) $J_i = J_j$ and (ii) either $X \oplus L_i = X^* \oplus L_j$ or $Y \oplus L_i = Y^* \oplus L_j$, where $K_i = J_i \parallel L_i$ and $K_j = J_j \parallel L_j$ are the keys of users i and j respectively as indicated by the transcript. In the real world these entries must be consistent, but it is unlikely in the ideal world.

If a transcript is not bad then we say that it's *good*.

PROBABILITY OF BAD TRANSCRIPTS. Let $\mathcal{T}_0$ be the random variable for the transcript in the ideal system. We now bound the probability that $\mathcal{T}_0$ is bad. For each $i \in \{1, 2\}$, let Bad_i be the set of transcripts that violates the i -th constraint of badness.

We first bound the probability that $\mathcal{T}_0 \in \text{Bad}_1$. Consider a pair of entries (prim, J, X, Y) and $(\text{eval}, i, X^*, Y^*)$. Recall that in the ideal world, J_i and L_i are uniformly random, independent of those entries. Thus the chance that $J = J_i$ and either $X^* = X \oplus L_i$ or $Y^* = Y \oplus L_i$ is at most $2/2^{k+n}$. Summing that over pq pairs of such entries,

$$\Pr[\mathcal{T}_0 \in \text{Bad}_1] \leq \frac{2pq}{2^{k+n}} .$$

We next bound the probability that $\mathcal{T}_0 \in \text{Bad}_2$. Consider a pair of entries (eval, i, X, Y) and $(\text{eval}, j, X^*, Y^*)$, with $i \neq j$. Recall that in the ideal world, J_i, J_j, L_i, L_j are uniformly random, independent of those entries. Thus the chance that $J_i = J_j$ and either $X \oplus L_i = X^* \oplus L_j$ or $Y \oplus L_i = Y^* \oplus L_j$ is at most $2/2^{k+n}$. Summing that over $\binom{q}{2} \leq \frac{q^2}{2}$ pairs of such entries,

$$\Pr[\mathcal{T}_0 \in \text{Bad}_2] \leq \frac{q^2}{2^{k+n}} .$$

Summing up,

$$\Pr[\mathcal{T}_0 \text{ is bad}] \leq \Pr[\mathcal{T}_0 \in \text{Bad}_1] + \Pr[\mathcal{T}_0 \in \text{Bad}_2] \leq \frac{2pq + q^2}{2^{k+n}} . \quad (3)$$

TRANSCRIPT RATIO. Fix a good transcript τ . For a key $J \in \{0, 1\}^k$, let $V(J)$ be the number of entries (prim, J, X, Y) , and let $U(J)$ be the number of entries (eval, i, X, Y) such that $J_i = J$. Suppose that the transcript τ contains exactly u users. Then in the ideal world,

$$\Pr[\mathcal{T}_0 = \tau] \leq 2^{-u(k+n)} \prod_{J \in \{0, 1\}^k} \prod_{i=0}^{V(J)-1} \frac{1}{2^n - i} \prod_{j=0}^{U(J)-1} \frac{1}{2^n - j} .$$

In the real world, since τ is good,

$$\Pr[\mathcal{T}_1 = \tau] = 2^{-u(k+n)} \prod_{J \in \{0, 1\}^k} \prod_{i=0}^{V(J)-1} \frac{1}{2^n - i} \prod_{j=0}^{U(J)-1} \frac{1}{2^n - V(J) - j} .$$

Hence

$$\frac{\Pr[\mathcal{T}_1 = \tau]}{\Pr[\mathcal{T}_0 = \tau]} \geq 1 . \quad (4)$$

WRAPPING UP. From Lemma 5 with $\epsilon = 0$, Eq. (3), and Eq. (4),

$$\text{Adv}_G^{\pm \text{prp}}(\mathcal{A}) \leq \frac{2pq + q^2}{2^{k+n}} .$$

as claimed.

D Proof of Theorem 2

Our proof relies on the Chi-Squared method of Dai, Hoang, and Tessaro [10], and the well-known Azuma's inequality, which we will review below.

THE CHI-SQUARED METHOD. Suppose that we want to bound the advantage of a computationally unbounded adversary $\mathcal{A}$ in distinguishing a “real” system $\mathbf{S}_1$ from an “ideal” system $\mathbf{S}_0$. Without loss of generality, assume that $\mathcal{A}$ is deterministic and makes exactly q queries. Since the adversary is deterministic, for any $i \leq q - 1$, the answers for the first i queries completely determine the first $i + 1$ queries. For a system $\mathbf{S} \in \{\mathbf{S}_1, \mathbf{S}_0\}$ and strings $z_1, \dots, z_i$, let $\text{ps}_{\mathbf{S}, \mathcal{A}}(z_1, \dots, z_i)$ denote the probability that the answers for the first i queries that $\mathcal{A}$ receives when interacting with $\mathbf{S}$ are $z_1, \dots, z_i$. If $\text{ps}_{\mathbf{S}, \mathcal{A}}(z_1, \dots, z_i) > 0$, let $\text{ps}_{\mathbf{S}, \mathcal{A}}(z_{i+1} \mid z_1, \dots, z_i)$ denote the conditional probability that

the answer for the $(i+1)$ -th query when $\mathcal{A}$ interacts with system $\mathbf{S}$ is z_{i+1} , given that the answers for the first i queries are $z_1, \dots, z_i$ respectively.

For each $\mathbf{Z} = (z_1, \dots, z_q)$, let $\mathbf{Z}_i = (z_1, \dots, z_i)$ and let $\mathbf{Z}_0$ be the empty string. We write $\mathbf{p}_{\mathbf{S}, \mathcal{A}}(\cdot \mid \mathbf{Z}_i)$ and $\mathbf{p}_{\mathbf{S}, \mathcal{A}}(\cdot \mid \mathbf{Z}_0)$ to refer to probabilities $\mathbf{p}_{\mathbf{S}, \mathcal{A}}(\cdot \mid z_1, \dots, z_i)$ and $\mathbf{p}_{\mathbf{S}, \mathcal{A}}(\cdot)$ respectively. We require that if $\mathbf{p}_{\mathbf{S}_1, \mathcal{A}}(\mathbf{Z}_i) > 0$ then so is $\mathbf{p}_{\mathbf{S}_0, \mathcal{A}}(\mathbf{Z}_i)$. For each $i \leq q$ and each vector $\mathbf{Z}_{i-1} = (z_1, \dots, z_{i-1})$, define

$$\chi^2(\mathbf{Z}_{i-1}) = \sum_{z_i} \frac{(\mathbf{p}_{\mathbf{S}_1, \mathcal{A}}(z_i \mid \mathbf{Z}_{i-1}) - \mathbf{p}_{\mathbf{S}_0, \mathcal{A}}(z_i \mid \mathbf{Z}_{i-1}))^2}{\mathbf{p}_{\mathbf{S}_0, \mathcal{A}}(z_i \mid \mathbf{Z}_{i-1})} ,$$

where the sum is taken over all z_i such that $\mathbf{p}_{\mathbf{S}_0, \mathcal{A}}(z_i \mid \mathbf{Z}_{i-1}) > 0$. Lemma 6 below bounds the statistical distance $\text{SD}(\mathbf{p}_{\mathbf{S}_0, \mathcal{A}}(\cdot), \mathbf{p}_{\mathbf{S}_1, \mathcal{A}}(\cdot))$ between $\mathbf{p}_{\mathbf{S}_0, \mathcal{A}}(\cdot)$ and $\mathbf{p}_{\mathbf{S}_1, \mathcal{A}}(\cdot)$, namely the best possible distinguishing advantage of $\mathcal{A}$ between $\mathbf{S}_1$ and $\mathbf{S}_0$.

Lemma 6 (The Chi-Squared Lemma). [10, Lemma 3] *Suppose whenever $\mathbf{p}_{\mathbf{S}_1, \mathcal{A}}(\mathbf{Z}_i) > 0$ then $\mathbf{p}_{\mathbf{S}_0, \mathcal{A}}(\mathbf{Z}_i) > 0$. Then*

$$\text{SD}(\mathbf{p}_{\mathbf{S}_0, \mathcal{A}}(\cdot), \mathbf{p}_{\mathbf{S}_1, \mathcal{A}}(\cdot)) \leq \left(\frac{1}{2} \sum_{i=1}^q \mathbf{E}[\chi^2(\mathbf{X}_{i-1})] \right)^{1/2} ,$$

where the expectation is taken over vectors $\mathbf{X}_{i-1}$ of the $i-1$ first answers sampled according to the interaction with $\mathbf{S}_1$.

AZUMA'S INEQUALITY. We now briefly recall the Azuma's inequality; a more detailed exposition can be found in a standard textbook of probability theory, for example, [25, Chapter 4.4].

Let $T_0, T_1, \dots, T_m$ be a sequence of random variables. We say that $(T_0, \dots, T_m)$ is a *martingale* if $\mathbf{E}[|T_i|] < \infty$ and $\mathbf{E}[T_i \mid T_1, \dots, T_{i-1}] = T_{i-1}$ for every $i \in \{1, \dots, m\}$. The following result is the well-known Azuma's inequality.

Lemma 7 (Azuma's inequality). *Let $T_0, \dots, T_m$ be a martingale. Suppose that there is a constant c that $|T_i - T_{i-1}| \leq c$ for every $i \in \{1, \dots, m\}$. Then for any $\lambda > 0$,*

$$\Pr[|T_m - T_0| \geq c\lambda\sqrt{m}] \leq e^{-\lambda^2/2} .$$

THE PROOF. Consider games P_1 – P_4 in Fig. 15. Game P_1 corresponds to game $\mathbf{G}_{\text{XOR}[E]}^{\text{prf}}$ with challenge bit $b = 1$, and game P_4 corresponds to game $\mathbf{G}_{\text{XOR}[E]}^{\text{prf}}$ with challenge bit $b = 0$. Game P_2 is similar to game P_1 , but it samples a truly random permutation π_i for each user i , and for each call to $E(K_i, \cdot)$, it calls $\pi_i(\cdot)$ instead. Game P_3 is similar to game P_4 , but instead of picking a truly random function f_i for each user i , it imposes a constraint that $f_i(X) \neq 0^n$ for every $x \in \{0, 1\}^{n-1}$.

To bound the gap between games P_1 and P_2 , we construct an adversary $\mathcal{B}$ attacking the PRP security of G as follows. Adversary $\mathcal{B}$ runs $\mathcal{A}$ and simulates game P_1 . However, for each call to $G(K_i, \cdot)$, it instead makes a corresponding query to its oracle $\text{EVAL}(i, \cdot)$. Thus

$$\Pr[P_1(\mathcal{A})] - \Pr[P_2(\mathcal{A})] = \text{Adv}_G^{\text{prp}}(\mathcal{B}) .$$

procedure INITIALIZE//Game P_1 $K_1, K_2, \dots \leftarrow \mathcal{K}$ procedure EVAL(i, M) $Y \leftarrow G(K_i, M \parallel 0) \oplus G(K_i, M \parallel 1)$ return Y	procedure INITIALIZE//Game P_2 $\pi_1, \pi_2, \dots \leftarrow \text{Perm}(n)$ procedure EVAL(i, M) $Y \leftarrow \pi_i(M \parallel 0) \oplus \pi_i(M \parallel 1)$ return Y
procedure INITIALIZE//Games P_3 $\text{Dom} \leftarrow \{0, 1\}^{n-1}$ $\text{Rng} \leftarrow \{0, 1\}^n \setminus \{0^n\}$ $f_1, f_2, \dots \leftarrow \text{Func}(\text{Dom}, \text{Rng})$ procedure EVAL(i, M) $Y \leftarrow f_i(M)$; return Y	procedure INITIALIZE//Games P_4 $\text{Dom} \leftarrow \{0, 1\}^{n-1}$ $\text{Rng} \leftarrow \{0, 1\}^n$ $f_1, f_2, \dots \leftarrow \text{Func}(\text{Dom}, \text{Rng})$ procedure EVAL(i, M) $Y \leftarrow f_i(M)$; return Y

Fig. 15: Games P_1 – P_4 in the proof of Theorem 2. The games share a common FINALIZE(b') that returns ($b' = 1$).

procedure EVAL(i, M) //Games $\boxed{P_3}$, P_4 $Y \leftarrow \{0, 1\}^n$ if $Y = 0^n$ then $\text{bad} \leftarrow \text{true}$; $Y \leftarrow \{0, 1\}^n \setminus \{0^n\}$ return Y
--

Fig. 16: Rewritten games P_3 and P_4 in the proof of Theorem 2. Procedure INITIALIZE is empty and thus omitted. Game P_3 contains the boxed statement but game P_4 does not.

For games P_2 – P_4 , we will consider even a computationally unbounded adversary $\mathcal{A}$, and thus we can assume that $\mathcal{A}$ is deterministic and never repeats a prior query. We claim that

$$\Pr[P_2(\mathcal{A})] - \Pr[P_3(\mathcal{A})] \leq \frac{\sqrt{50n} \cdot q}{2^n} ;$$

the proof is postponed until further below. For games P_3 and P_4 , due to the assumption that $\mathcal{A}$ does not repeat a prior query, instead of sampling functions f_i eagerly, we can sample them lazily, and thus the two games can be rewritten as shown in Fig. 16. In particular, in game P_4 , each query will receive a truly random answer, and in game P_3 , each query will receive a random, non-zero answer. The two games are identical until the flag **bad** is set, and thus from the Fundamental Lemma of Game Playing [3],

$$\Pr[P_3(\mathcal{A})] - \Pr[P_4(\mathcal{A})] \leq \Pr[P_4 \text{ sets bad}] .$$

If $\mathcal{A}$ makes q queries then the chance that one of the truly random answers becomes 0^n is at most $q/2^n$. Hence

$$\Pr[P_3(\mathcal{A})] - \Pr[P_4(\mathcal{A})] \leq \frac{q}{2^n} .$$

Summing up,

$$\begin{aligned} \text{Adv}_{\text{XOR}[G]}^{\text{prf}}(\mathcal{A}) &= \Pr[P_1(\mathcal{A})] - \Pr[P_4(\mathcal{A})] \\ &= \sum_{i=1}^3 \Pr[P_i(\mathcal{A})] - \Pr[P_{i+1}(\mathcal{A})] \leq \text{Adv}_G^{\text{prp}}(\mathcal{B}) + \frac{8\sqrt{n} \cdot q}{2^n} . \end{aligned}$$

We now bound the gap between P_2 and P_3 via the Chi-Squared method. Let $\mathbf{S}_1$ be the system that implements game P_2 , and $\mathbf{S}_0$ be the system that implements game P_3 . Without loss of generality, suppose that $\mathcal{A}$ makes exactly q queries. Let $\mathbf{X} = (X_1, \dots, X_q)$ be the random variable for the q answers in $\mathbf{S}_1$, and let $\mathbf{X}_i = (X_1, \dots, X_i)$ for every $i \leq q$. Fix $i \leq q$ and fix $x \in \{0, 1\}^n \setminus \{0^n\}$. Let $Y_{i,x}$ be the following random variable. If $\mathbf{X}_{i-1}$ takes values $(z_1, \dots, z_{i-1})$ then $Y_{i,x}$ takes the value $\mathbf{ps}_{1,A}(x \mid z_1, \dots, z_{i-1})$. Recall that

$$\chi^2(\mathbf{X}_{i-1}) = \sum_{x \in \{0,1\}^n \setminus \{0^n\}} \frac{(Y_{i,x} - 1/(2^n - 1))^2}{1/(2^n - 1)} \leq \sum_{x \in \{0,1\}^n \setminus \{0^n\}} 2^n \cdot (Y_{i,x} - 1/(2^n - 1))^2. \quad (5)$$

We claim that

$$\mathbf{E}\left[\left(Y_{i,x} - \frac{1}{2^n - 1}\right)^2\right] \leq \frac{100in}{2^{4n}}. \quad (6)$$

This claim will be justified later. From Eq. (5) and Eq. (6),

$$\mathbf{E}[\chi^2(\mathbf{X}_{i-1})] \leq \sum_{x \in \{0,1\}^n \setminus \{0^n\}} 2^n \cdot \mathbf{E}[(Y_{i,x} - 1/(2^n - 1))^2] \leq \frac{100in}{2^{2n}}.$$

By the Chi-Squared Lemma,

$$\begin{aligned} (\Pr[P_2(\mathcal{A})] - \Pr[P_3(\mathcal{A})])^2 &\leq \text{SD}(\mathbf{ps}_{0,A}(\cdot), \mathbf{ps}_{1,A}(\cdot))^2 \\ &\leq \frac{1}{2} \sum_{i=1}^q \mathbf{E}[\chi^2(\mathbf{X}_{i-1})] \leq \frac{1}{2} \sum_{i=1}^q \frac{100in}{2^{2n}} \leq \frac{50nq^2}{2^{2n}}. \end{aligned}$$

We now will justify Eq. (6) above. In the real system $\mathbf{S}_1$, the first $i-1$ queries lead to $2(i-1)$ random-permutation calls, and let $\mathbf{V} = (V_1, \dots, V_{2i-2})$ be the list of the outputs of those calls. For $v \in (\{0, 1\}^n)^{2i-2}$, we write $\mathbf{ps}_{1,A}(x \mid \mathbf{V} = v)$ to denote the conditional probability that the i -th answer is x , given that $\mathbf{V} = v$. Let W be the random variable that if $\mathbf{V}$ takes value v then W takes value $\mathbf{ps}_{1,A}(x \mid \mathbf{V} = v)$. Since $\mathbf{V}$ uniquely determines $\mathbf{X}_{i-1}$,

$$Y_{i,x} = \mathbf{E}[W \mid \mathbf{X}_{i-1}].$$

From the fact that the function $\rho(x) = (x - 1/(2^n - 1))^2$ is convex, using Jensen's inequality for conditional expectation,

$$\begin{aligned} \left(Y_{i,x} - \frac{1}{2^n - 1}\right)^2 &= \rho(\mathbf{E}[W \mid \mathbf{X}_{i-1}]) \\ &\leq \mathbf{E}[\rho(W) \mid \mathbf{X}_{i-1}] = \mathbf{E}\left[\left(W - \frac{1}{2^n - 1}\right)^2 \mid \mathbf{X}_{i-1}\right]. \end{aligned}$$

Taking expectation of both sides gives us

$$\mathbf{E}\left[\left(Y_{i,x} - \frac{1}{2^n - 1}\right)^2\right] \leq \mathbf{E}\left[\left(W - \frac{1}{2^n - 1}\right)^2\right].$$

What remains is to prove that

$$\mathbf{E}\left[\left(W - \frac{1}{2^n - 1}\right)^2\right] \leq \frac{100in}{2^{4n}}.$$

Since $(W - 1/(2^n - 1))^2 \leq 1$, it suffices to prove that

$$\Pr\left[\left(W - \frac{1}{2^n - 1}\right)^2 \geq \frac{99in}{2^{4n}}\right] \leq \frac{in}{2^{4n}} . \quad (7)$$

THE SINGLE-USER CASE. It is instructive to first consider the case that the adversary $\mathcal{A}$ targets only a single user. In this case, we actually can give a much stronger concentration bound on W . In particular, we will show that

$$\Pr\left[\left(W - \frac{1}{2^n - 1}\right)^2 \geq \frac{99in}{2^{4n}}\right] \leq 2^{-5n} . \quad (8)$$

Note that in this single-user setting, $V_1, \dots, V_{2i-2}$ are sampled uniformly without replacement from $\{0, 1\}^n$. For each $j \leq 2i - 2$, let B_j be the Bernoulli random variable such that $B_j = 1$ if and only if $V_j \in \{V_1 \oplus x, \dots, V_{j-1} \oplus x\}$. Let $D = 2(B_1 + \dots + B_{2i-2})$. We will use the following result that is implicit in the work of Dai, Hoang, and Tessaro [10].

Lemma 8. [10] Define $B_1, \dots, B_i$, and W and D as above. Then

$$\left(W - \frac{1}{2^n - 1}\right)^2 \leq \frac{3(D - 4(i-1)^2/2^n)^2 + 12(i-1)^2/2^{2n}}{2^{4n}} .$$

Moreover,

$$\mathbf{E}[B_i \mid B_1, \dots, B_{i-1}] = \frac{i-1 - 2(B_1 + \dots + B_{i-1})}{2^n - (i-1)} .$$

We now prove that for every $j \leq 2i - 2$, with probability at least $1 - j/e^{4.5n} \geq 1 - 2^{-5n}$,

$$|B_1 + \dots + B_j - j^2/2^{n+1}| \leq 4\sqrt{jn} .$$

Combining this for $j = 2i - 2$ and Lemma 8 leads to Eq. (8). We prove this claim by induction on j . The base case $j = 1$ obviously holds, since $B_1 = 0$. Now, suppose that the claim holds for $j - 1$, we shall prove that it holds for j as well. Let $T_0 = 0$ and $T_r = T_{r-1} + B_r - \mathbf{E}[B_r \mid B_1, \dots, B_{r-1}]$ for every $1 \leq r \leq j$. Since $B_1, \dots, B_{r-1}$ completely determine $T_1, \dots, T_{r-1}$,

$$\begin{aligned} \mathbf{E}[T_r \mid T_1, \dots, T_{r-1}] &= \mathbf{E}[T_{r-1} + B_r - \mathbf{E}[B_r \mid B_1, \dots, B_{r-1}] \mid T_1, \dots, T_{r-1}] \\ &= T_{r-1} + \mathbf{E}[B_r \mid T_1, \dots, T_{r-1}] - \mathbf{E}[B_r \mid T_1, \dots, T_{r-1}] = T_{r-1} . \end{aligned}$$

Moreover, for every $r \leq j$,

$$|T_r - T_{r-1}| = |B_r - \mathbf{E}[B_r \mid B_1, \dots, B_{r-1}]| \leq \max\{B_r, \mathbf{E}[B_r \mid B_1, \dots, B_{r-1}]\} \leq 1 ,$$

where the first inequality is due to the fact that $B_r \geq 0$, and the second inequality is due to the fact that $B_r \leq 1$. Thus $|T_r| \leq r$, and therefore $\mathbf{E}[|T_r|] < \infty$. Hence $(T_0, \dots, T_j)$ is a martingale. Using Azuma's inequality with $c = 1$, $\lambda = 3\sqrt{n}$, and $m = j$,

$$\Pr[|T_j - T_0| \geq 3\sqrt{nj}] \leq e^{-4.5n} . \quad (9)$$

Now suppose that $|T_j - T_0| \leq 3\sqrt{nj}$ and for every $r \leq j-1$,

$$|B_1 + \dots + B_r - r^2/2^{n+1}| \leq 4\sqrt{rn} . \quad (10)$$

From Eq. (9) and the induction hypothesis, this happens with probability at least $1 - j/e^{4.5n}$. We now show that

$$|B_1 + \dots + B_j - j^2/2^{n+1}| \leq 4\sqrt{jn}$$

with (conditional) probability 1, justifying the inductive case. From Lemma 8, for every $r \leq j$,

$$\mathbf{E}[B_r \mid B_1, \dots, B_{r-1}] = \frac{r-1-2(B_1 + \dots + B_{r-1})}{2^n - (r-1)} . \quad (11)$$

Thus, from Eq. (11), for every $r \leq j$,

$$\begin{aligned} \left| \mathbf{E}[B_r \mid B_1, \dots, B_{r-1}] - (r-1)/2^n \right| &= \left| \frac{r-1-2(B_1 + \dots + B_{r-1})}{2^n - (r-1)} - (r-1)/2^n \right| \\ &= \left| \frac{-2(B_1 + \dots + B_{r-1} - (r-1)^2/2^{n+1})}{2^n - (r-1)} \right| \\ &\leq \frac{8\sqrt{(r-1)n}}{2^n - (r-1)} \leq \frac{8\sqrt{(r-1)n}}{2^n - j} , \end{aligned}$$

where the second last inequality is due to the induction hypothesis. Since

$$T_j - T_0 = \sum_{r=1}^j B_r - \mathbf{E}[B_r \mid B_1, \dots, B_{r-1}],$$

we have

$$\begin{aligned} \left| B_1 + \dots + B_j - j(j-1)/2^{n+1} \right| &= \left| \sum_{r=1}^j B_r - (r-1)/2^n \right| \\ &= \left| T_j - T_0 + \sum_{r=1}^j (\mathbf{E}[B_r \mid B_1, \dots, B_{r-1}] - (r-1)/2^n) \right| \\ &\leq |T_j - T_0| + \sum_{r=1}^j \left| \mathbf{E}[B_r \mid B_1, \dots, B_{r-1}] - (r-1)/2^n \right| \leq 3\sqrt{jn} + \sum_{r=1}^j \frac{8\sqrt{(r-1)n}}{2^n - j} . \end{aligned} \quad (12)$$

On the other hand,

$$\begin{aligned} \sum_{r=1}^j \frac{8\sqrt{(r-1)n}}{2^n - j} &= \frac{8j\sqrt{jn}}{2^n - j} \cdot \frac{1}{j} \sum_{r=1}^j \sqrt{(r-1)/j} \\ &\leq \frac{8j\sqrt{jn}}{2^n - j} \cdot \int_0^1 \sqrt{x} dx = \frac{16j\sqrt{jn}}{3 \cdot (2^n - j)} \leq \sqrt{jn} - \frac{j}{2^{n+1}} , \end{aligned} \quad (13)$$

where the last inequality is due to the fact that $n \geq 8$ and $j \leq \frac{2^n}{128}$. From Eq. (12) and Eq. (13),

$$\begin{aligned} \left| B_1 + \dots + B_j - j^2/2^{n+1} \right| &\leq \left| B_1 + \dots + B_j - j(j-1)/2^{n+1} \right| + \frac{j}{2^n} \\ &\leq 3\sqrt{jn} + \sqrt{jn} - \frac{j}{2^n} + \frac{j}{2^n} = 4\sqrt{jn} . \end{aligned}$$

<pre> procedure INITIALIZE//Games G_1, G_2 $K_1, K_2, \dots \leftarrow \mathcal{K}; J_1, J_2, \dots \leftarrow 0$ $f_1, f_2, \dots \leftarrow \text{Func}(\mathcal{N}, \{0, 1\}^k \times \{0, 1\}^\ell)$ procedure Enc.init(i, N) $J_i \leftarrow J_i + 1; j \leftarrow J_i$ $(L, P) \leftarrow \text{KD}(K_i, N)$ //For game G_1 only $(L, P) \leftarrow f_i(N)$ //For game G_2 only $S_{i,j} \leftarrow (L, P, N, 1); \text{return } J_i$ procedure Enc.next(i, j, A, M) if $S_{i,j} = \perp$ then return $\perp$ $(L, P, N, r) \leftarrow S_{i,j}; C \leftarrow \mathbf{E}_L(\langle P, r, 0 \rangle, A, M)$ $S_{i,j} \leftarrow (L, P, N, r + 1); \text{return } C$ procedure Enc.last(i, j, A, M) if $S_{i,j} = \perp$ then return $\perp$ $(L, P, N, r) \leftarrow S_{i,j}$ $C \leftarrow \mathbf{E}_L(\langle P, r, 1 \rangle, A, M); S_{i,j} \leftarrow \perp$ return C </pre>	<pre> procedure Dec(i, N, A, C, a) if $A \neq C$ then return false $m \leftarrow C$ $(L, P) \leftarrow \text{KD}(K_i, N)$ //For game G_1 only $(L, P) \leftarrow f_i(N)$ //For game G_2 only for $r \leftarrow 1$ to I do $j \leftarrow \mathcal{I}[r]; \text{if } j > m \text{ or } j < 1 \text{ then return false}$ if $(a = 0 \text{ or } j < m)$ then if $\mathbf{D}_L(\langle P, j, 0 \rangle, A[j], C[j]) = \perp$ then return false elseif $\mathbf{D}_L(\langle P, m, 1 \rangle, A[m], C[m]) = \perp$ then return false return true procedure FINALIZE(b') return $(b' = 1)$ </pre>
---	---

Fig. 17: Games G_1 and G_2 in the proof of Theorem 3. Each game maintains an internal procedure LOOK.

THE MULTI-USER CASE. We now prove Eq. (7) for the general multi-user setting. Let Bad be the event that $(W - 1/(2^n - 1))^2 \geq 99in/2^{4n}$. One can think of the adversary as playing a (multi-user) game; it wins if Bad happens. Here the adversary has the adaptivity to pick the user U for the i -th query, and also the number $Q \in \{0, \dots, i - 1\}$ of queries, among the first $i - 1$ queries, for this user. By reindexing the user, we can assume that $U \in \{1, \dots, i\}$. Note that Bad only depends the queries and answers of user U . Hence each fixed choice of (U, Q) corresponds to a single-user game in which Bad happens with probability at most 2^{-5n} , thanks to Eq. (8). The adaptivity at best lets the adversary play i^2 single-user games simultaneously; if the adversary wins some of those games, it wins its multi-user game. From the Union Bound, with probability at most $i^2/2^{5n} \leq i/2^{4n}$, the adversary can win the multi-user game.

E Proof of Theorem 3

Consider games G_1 – G_4 in Fig. 17 and Fig. 18. Game G_1 corresponds to game $\mathbf{G}_{\text{SE2}[\text{KD}, \Pi]}^{\text{noae2}}(\mathcal{A})$ with challenge bit $b = 1$. In game G_1 , each user i has a corresponding key K_i , but in game G_2 , we sample a uniformly random function $f_i : \mathcal{N} \rightarrow \{0, 1\}^k \times \{0, 1\}^\ell$, and for each call to $\text{KD}(K_i, \cdot)$, we make a corresponding call to $f_i(\cdot)$ instead.

Game G_3 is similar to game G_2 , but in each encryption query for user i and nonce N that results in a call $C \leftarrow \mathbf{E}_L(N^*, A, M)$, we store the tuple (i, N, N^*, A, C) in a table. On the other hand, each time we are supposed to check if $\mathbf{D}_L(N^*, A, C) \neq \perp$ for a user i under nonce N , if there is already a prior tuple (i, N, N^*, A, C) then we simply return true. Otherwise we will indeed call $\mathbf{D}_L(N^*, A, C)$.

Game G_4 is similar to game G_3 , but for each call $C \leftarrow \mathbf{E}_L(N^*, A, M)$, we instead pick a truly random C of the same length. Moreover, for each call $\mathbf{D}_L(N^*, A, C)$, we instead use $\perp$ as the answer. Thanks to the restriction on the behavior of the adversary, any call to the Dec oracle must eventually end up with a call to $\mathbf{D}$ instead of looking up prior tuples from encryption queries.

<pre> procedure INITIALIZE//Games G_3, G_4 $J_1, J_2, \dots \leftarrow 0$ $f_1, f_2, \dots \leftarrow \text{sFunc}(\mathcal{N}, \{0, 1\}^k \times \{0, 1\}^\ell)$ procedure Enc.init(i, N) $J_i \leftarrow J_i + 1; j \leftarrow J_i$ $(L, P) \leftarrow f_i(R); S_{i,j} \leftarrow (L, P, N, 1)$ return J_i procedure Enc.next(i, j, A, M) if $S_{i,j} = \perp$ then return $\perp$ $(L, P, N, r) \leftarrow S_{i,j}; N^* \leftarrow \langle P, r, 0 \rangle$ $C \leftarrow \mathbf{E}_L(N^*, A, M)$ $C \leftarrow \text{s} \{0, 1\}^{ C }$ //For game G_4 only $\text{Tab}[i, R, N^*, A, C] \leftarrow \text{true}$ $S_{i,j} \leftarrow (L, P, N, r + 1);$ return C procedure Enc.last(i, j, A, M) if $S_{i,j} = \perp$ then return $\perp$ $(L, P, N, r) \leftarrow S_{i,j}; N^* \leftarrow \langle P, r, 1 \rangle$ $C \leftarrow \mathbf{E}_L(N^*, A, M)$ $C \leftarrow \text{s} \{0, 1\}^{ C }$ //For game G_4 only $\text{Tab}[i, R, N^*, A, C] \leftarrow \text{true}; S_{i,j} \leftarrow \perp$ return C </pre>	<pre> procedure Dec($i, N, \mathbf{A}, \mathbf{C}, a$) if $\mathbf{A} \neq \mathbf{C}$ then return false $m \leftarrow \mathbf{C} ; (L, P) \leftarrow f_i(N)$ for $r \leftarrow 1$ to $\mathcal{I}$ do $j \leftarrow \mathcal{I}[r];$ if $j > m$ or $j < 1$ then return false if $(a = 0$ or $j < m)$ then if LOOK($i, N, \langle P, j, 0 \rangle, \mathbf{A}[j], \mathbf{C}[j]$) = false then return false elseif LOOK($i, N, \langle P, m, 1 \rangle, \mathbf{A}[m], \mathbf{C}[m]$) = false then return false return true private procedure LOOK(i, N, N^*, A, C) if $\text{Tab}[i, N, N^*, A, C] \neq \perp$ then return true $(L, P) \leftarrow f_i(N); M \leftarrow \mathbf{D}_L(N^*, A, C)$ $M \leftarrow \perp$ //For game G_4 only return $(M \neq \perp)$ procedure FINALIZE(b') return $(b' = 1)$ </pre>
--	--

Fig. 18: **Games G_3 and G_4 in the proof of Theorem 3.** Each game maintains an internal procedure LOOK.

As a result, any Dec query will result in a false-answer. Hence game G_4 coincides with game $\mathbf{G}_{\text{SE2}[\text{KD}, \Pi]}^{\text{noae2}}(\mathcal{A})$ of challenge bit $b = 0$, and thus

$$\text{Adv}_{\text{SE2}[\text{KD}, \Pi]}^{\text{noae2}}(\mathcal{A}) = \Pr[G_1(\mathcal{A})] - \Pr[G_4(\mathcal{A})] .$$

We now bound the gap between each pair of consecutive games. Consider the following adversary $\mathcal{B}'$ attacking the (muti-user) PRF security of KD. It runs $\mathcal{A}$ and simulates game G_1 , but for each call to $\text{KD}(K_i, \cdot)$, it instead uses its oracle $\text{EVAL}(i, \cdot)$. Thus

$$\Pr[G_1(\mathcal{A})] - \Pr[G_2(\mathcal{A})] = \text{Adv}_{\text{KD}}^{\text{prf}}(\mathcal{B}') .$$

Next, in game G_3 , for each table look-up $\text{LOOK}(i, N, N^*, A, M)$ with $N^* = \langle P, j, a \rangle$, the triple (i, N, N^*) uniquely determines the user i , nonce N , and counter j . Thus the table look-ups do not return false positives. Thanks to the correctness of the conventional AE scheme Π ,

$$\Pr[G_2(\mathcal{A})] = \Pr[G_3(\mathcal{A})] .$$

Finally, consider the following adversary $\mathcal{B}$ attacking Π . It runs $\mathcal{A}$ and simulates game G_3 , but with the following differences. First, instead of eagerly sampling a function f_i for each user i , adversary $\mathcal{B}$ lazily implements f_i . That is, each time it has to evaluate $f_i(N)$, if this is the first time it has to do this evaluation, it picks $L \leftarrow \text{s} \{0, 1\}^k$ and $P \leftarrow \text{s} \{0, 1\}^\ell$, stores the triple $(\langle i, N \rangle, L, P)$, and returns (L, P) as the answer, where $\langle i, N \rangle$ is a unique encoding of (i, N) as an integer. Otherwise, if there is already some pair $(\langle i, N \rangle, L, P)$ then it simply returns (L, P) as the answer. Next, for each call

$\mathbf{E}_L(N^*, A, M)$ for a user i under nonce N , it instead calls $\text{ENC}(\langle i, N \rangle, N^*, A, M)$. Likewise, for each call $\mathbf{D}_L(N^*, A, C)$ for a user i under nonce N , it calls $\text{DEC}(\langle i, N \rangle, N^*, A, C)$.

Thanks to the table look-ups in game G_3 , adversary $\mathcal{B}$ does not make prohibited queries to the decryption oracle. Moreover, for each encryption query $\text{ENC}(\langle i, N \rangle, N^*, A, M)$ with $N^* = \langle P, j, a \rangle$, the pair $(\langle i, N \rangle, N^*)$ uniquely determines the user i , nonce N , and counter j . Hence $\mathcal{B}$ is nonce-respecting, as long as $\mathcal{A}$ is nonce-respecting. Next, view each encryption of a (partial) stream as throwing a ball into bin P , where P is the corresponding nonce prefix. The throws are uniformly random. From Lemma 2, $\mathcal{B}$ is $\max\{4\ell, 4q/2^\ell\}$ -repeating with probability at least $1 - 2^{-\ell}$. On the other hand,

$$\Pr[G_3(\mathcal{A})] - \Pr[G_4(\mathcal{A})] = \text{Adv}_{\Pi}^{\text{ae}}(\mathcal{B}) .$$

Summing up

$$\begin{aligned} \text{Adv}_{\text{SE2}[\text{KD}, \Pi]}^{\text{noae2}}(\mathcal{A}) &= \Pr[G_1(\mathcal{A})] - \Pr[G_4(\mathcal{A})] \\ &= \sum_{i=1}^3 \Pr[G_i(\mathcal{A})] - \Pr[G_{i+1}(\mathcal{A})] = \text{Adv}_{\text{KD}}^{\text{prf}}(\mathcal{B}') + \text{Adv}_{\Pi}^{\text{ae}}(\mathcal{B}) . \end{aligned}$$

F Proof of Theorem 4

Consider games G_1 – G_4 in Fig. 19 and Fig. 20. Game G_1 corresponds to game $\mathbf{G}_{\text{SE3}[\text{KD}, \Pi]}^{\text{noae2}}(\mathcal{A})$ with challenge bit $b = 1$. In game G_1 , each user i has a corresponding key K_i , but in game G_2 , we sample a uniformly random function $f_i : \{0, 1\}^s \rightarrow \{0, 1\}^k \times \{0, 1\}^\ell$, and for each call to $\text{KD}(K_i, \cdot)$, we make a corresponding call to $f_i(\cdot)$ instead.

Game G_3 is similar to game G_2 , but in each encryption query for user i and nonce $N = (R, P)$ that results in a call $C \leftarrow \mathbf{E}_L(N^*, A, M)$, we store the tuple (i, R, N^*, A, C) in a table. On the other hand, each time we are supposed to check if $\mathbf{D}_L(N^*, A, C) \neq \perp$ for a user i under nonce $N = (R, P)$, if there is already a prior tuple (i, R, N^*, A, C) then we simply return **true**. Otherwise we will indeed call $\mathbf{D}_L(N^*, A, C)$.

Game G_4 is similar to game G_3 , but for each call $C \leftarrow \mathbf{E}_L(N^*, A, M)$, we instead pick a truly random C of the same length. Moreover, for each call $\mathbf{D}_L(N^*, A, C)$, we instead use $\perp$ as the answer. Thanks to the restriction on the behavior of the adversary, any call to the Dec oracle must eventually end up with a call to $\mathbf{D}$ instead of looking up prior tuples from encryption queries. As a result, any Dec query will result in a false-answer. Hence game G_4 coincides with game $\mathbf{G}_{\text{SE3}[\text{KD}, \Pi]}^{\text{noae2}}(\mathcal{A})$ of challenge bit $b = 0$, and thus

$$\text{Adv}_{\text{SE3}[\text{KD}, \Pi]}^{\text{noae2}}(\mathcal{A}) = \Pr[G_1(\mathcal{A})] - \Pr[G_4(\mathcal{A})] .$$

We now bound the gap between each pair of consecutive games. Consider the following adversary $\mathcal{B}'$ attacking the (muti-user) PRF security of KD. It runs $\mathcal{A}$ and simulates game G_1 , but for each call to $\text{KD}(K_i, \cdot)$, it instead uses its oracle $\text{EVAL}(i, \cdot)$. Thus

$$\Pr[G_1(\mathcal{A})] - \Pr[G_2(\mathcal{A})] = \text{Adv}_{\text{KD}}^{\text{prf}}(\mathcal{B}') .$$

<pre> procedure INITIALIZE//Games G_1, G_2 $K_1, K_2, \dots \leftarrow \mathcal{K}$; $J_1, J_2, \dots \leftarrow 0$ $f_1, f_2, \dots \leftarrow \text{Func}(\{0, 1\}^s, \{0, 1\}^k \times \{0, 1\}^\ell)$ procedure Enc.init(i, N) $(R, P) \leftarrow N$; $J_i \leftarrow J_i + 1$; $j \leftarrow J_i$ $(L, X) \leftarrow \text{KD}(K_i, R)$ //For game G_1 only $(L, X) \leftarrow f_i(R)$ //For game G_2 only $P^* \leftarrow P \oplus X$; $S_{i,j} \leftarrow (L, N, P^*, 1)$; return J_i procedure Enc.next(i, j, A, M) if $S_{i,j} = \perp$ then return $\perp$ $(L, N, P^*, r) \leftarrow S_{i,j}$; $C \leftarrow \mathbf{E}_L(\langle P^*, r, 0 \rangle, A, M)$ $S_{i,j} \leftarrow (L, N, P^*, r + 1)$; return C procedure Enc.last(i, j, A, M) if $S_{i,j} = \perp$ then return $\perp$ $(L, N, P^*, r) \leftarrow S_{i,j}$; $C \leftarrow \mathbf{E}_L(\langle P^*, r, 1 \rangle, A, M)$ $S_{i,j} \leftarrow \perp$; return C </pre>	<pre> procedure Dec($i, N, A, C, \mathcal{I}, a$) if $A \neq C$ then return false $(R, P) \leftarrow N$; $m \leftarrow C$ $(L, X) \leftarrow \text{KD}(K_i, R)$ //For game G_1 only $(L, X) \leftarrow f_i(R)$ //For game G_2 only $P^* \leftarrow P \oplus X$ for $r \leftarrow 1$ to $\mathcal{I}$ do $j \leftarrow \mathcal{I}[r]$; if $j > m$ or $j < 1$ then return false if $(a = 0 \text{ or } j < m)$ then if $(\mathbf{D}_L(\langle P^*, j, 0 \rangle, A[j], C[j]) = \perp)$ then return false elseif $\mathbf{D}_L(\langle P^*, m, 1 \rangle, A[m], C[m]) = \perp$ then return false return true procedure FINALIZE(b') return $(b' = 1)$ </pre>
---	---

Fig. 19: Games G_1 and G_2 in the proof of Theorem 4. Each game maintains an internal procedure LOOK.

<pre> procedure INITIALIZE//Games G_3, G_4 $J_1, J_2, \dots \leftarrow 0$ $f_1, f_2, \dots \leftarrow \text{Func}(\{0, 1\}^s, \{0, 1\}^k \times \{0, 1\}^\ell)$ procedure Enc.init(i, N) $(R, P) \leftarrow N$; $J_i \leftarrow J_i + 1$; $j \leftarrow J_i$ $(L, X) \leftarrow f_i(R)$; $P^* \leftarrow P \oplus X$ $S_{i,j} \leftarrow (L, N, P^*, 1)$; return J_i procedure Enc.next(i, j, A, M) if $S_{i,j} = \perp$ then return $\perp$ $(L, N, P^*, r) \leftarrow S_{i,j}$; $(R, P) \leftarrow N$ $N^* \leftarrow \langle P^*, r, 0 \rangle$; $C \leftarrow \mathbf{E}_L(N^*, A, M)$ $C \leftarrow \{0, 1\}^{ C }$ //For game G_4 only $\text{Tab}[i, R, N^*, A, C] \leftarrow \text{true}$ $S_{i,j} \leftarrow (L, N, P^*, r + 1)$; return C procedure Enc.last(i, j, A, M) if $S_{i,j} = \perp$ then return $\perp$ $(L, N, r) \leftarrow S_{i,j}$; $(R, P) \leftarrow N$ $N^* \leftarrow \langle P, r, 1 \rangle$; $C \leftarrow \mathbf{E}_L(N^*, A, M)$ $C \leftarrow \{0, 1\}^{ C }$ //For game G_4 only $\text{Tab}[i, R, N^*, A, C] \leftarrow \text{true}$; $S_{i,j} \leftarrow \perp$ return C </pre>	<pre> procedure Dec(i, N, A, C, a) if $A \neq C$ then return false $(R, P) \leftarrow N$; $m \leftarrow C$ for $r \leftarrow 1$ to $\mathcal{I}$ do $j \leftarrow \mathcal{I}[r]$; if $j > m$ or $j < 1$ then return false if $(a = 0 \text{ or } j < m)$ then if $\text{LOOK}(i, R, \langle P, j, 0 \rangle, A[j], C[j]) = \text{false}$ then return false elseif $\text{LOOK}(i, R, \langle P, m, 1 \rangle, A[m], C[m]) = \text{false}$ then return false return true private procedure LOOK(i, R, N^*, A, C) if $\text{Tab}[i, R, N^*, A, C] \neq \perp$ then return true $(L, X) \leftarrow f_i(R)$; $M \leftarrow \mathbf{D}_L(N^*, A, C)$ $M \leftarrow \perp$ //For game G_4 only return $(M \neq \perp)$ procedure FINALIZE(b') return $(b' = 1)$ </pre>
--	---

Fig. 20: Games G_3 and G_4 in the proof of Theorem 4. Each game maintains an internal procedure LOOK.

Next, in game G_3 , for each table look-up $\text{LOOK}(i, R, N^*, A, C)$ with $N^* = \langle P, j, a \rangle$, the triple (i, R, N^*) uniquely determines the user i , nonce $N = (R, P)$, and counter j . Since $\mathcal{A}$ is nonce-respecting, the table look-ups do not return false positives. Thanks to the correctness of the con-

ventional AE scheme Π ,

$$\Pr[G_2(\mathcal{A})] = \Pr[G_3(\mathcal{A})] .$$

Finally, consider the following adversary $\mathcal{B}$ attacking Π . It runs $\mathcal{A}$ and simulates game G_3 , but with the following differences.

- First, instead of eagerly sampling a function f_i for each user i , adversary $\mathcal{B}$ lazily implements f_i . That is, each time it has to evaluate $f_i(R)$, if this is the first time it has to do this evaluation, it picks $L \leftarrow_{\$} \{0, 1\}^k$ and $X \leftarrow_{\$} \{0, 1\}^\ell$, stores the pair $(\langle i, R \rangle, L, X)$, and returns (L, X) as the answer, where $\langle i, R \rangle$ is a unique encoding of (i, R) as an integer. Otherwise, if there is already some pair $(\langle i, R \rangle, L, X)$ then it simply returns (L, X) as the answer.
- Next, for each call $\mathbf{E}_L(N^*, A, M)$ for a user i under nonce $N = (R, P)$, it instead calls $\text{ENC}(\langle i, R \rangle, N^*, A, M)$. Likewise, for each call $\mathbf{D}_L(N^*, A, C)$ for a user i under nonce $N = (R, P)$, it calls $\text{DEC}(\langle i, R \rangle, N^*, A, C)$.
- Finally, $\mathcal{B}$ keeps track of how many times an effective nonce prefix P^* appears among (partial) encrypted streams. If there is some P^* that appears for $d = \max\{4\ell, 4q/2^\ell\}$ streams then it terminates immediately, outputting 1. This step can be implemented in linear time via a hash function. This termination ensures that $\mathcal{B}$ is d -repeating.

Thanks to the table look-ups in game G_3 , adversary $\mathcal{B}$ does not make prohibited queries to the decryption oracle. We claim that for each encryption query $\text{ENC}(\langle i, R \rangle, N^*, A, M)$ with $N^* = \langle P^*, j, a \rangle$, the pair $(\langle i, R \rangle, N^*)$ uniquely determines the user i and nonce $N = (R, P)$ and counter j . This implies $\mathcal{B}$ is nonce-respecting, as long as $\mathcal{A}$ is nonce-respecting. To justify this claim, assume to the contrary that there are two encryption queries that end up with the same pair $(\langle i, R \rangle, N^*)$, with $N^* = \langle P^*, j, a \rangle$. Let $(L, X) \leftarrow f_i(R)$, and let $P \leftarrow P^* \oplus X$. These two queries belong to the same user i , have the same nonce $N = (R, P)$, and use the same counter j , which leads to a contradiction if $\mathcal{A}$ is nonce-respecting.

Let c be the challenge bit of game $\mathbf{G}_\Pi^{\text{ae}}(\mathcal{B})$. Suppose that $c = 1$. If $\mathcal{B}$ does not terminate prematurely then it perfectly simulates game $G_3(\mathcal{A})$. Otherwise, it will always output 1, which is the correct answer. Hence

$$\Pr[\mathbf{G}_\Pi^{\text{ae}}(\mathcal{B}) \Rightarrow \text{true} \mid c = 1] \geq \Pr[G_3(\mathcal{A})] .$$

We claim that

$$\Pr[\mathbf{G}_\Pi^{\text{ae}}(\mathcal{B}) \Rightarrow \text{false} \mid c = 0] \leq \Pr[G_4(\mathcal{A})] + 2^{-\ell} . \quad (14)$$

Subtracting, we get

$$\text{Adv}_\Pi^{\text{ae}}(\mathcal{B}) \geq \Pr[G_3(\mathcal{A})] - \Pr[G_4(\mathcal{A})] - 2^{-\ell} .$$

Summing up,

$$\begin{aligned} \text{Adv}_{\mathbf{SE3}[\text{KD}, \Pi]}^{\text{noae2}}(\mathcal{A}) &= \Pr[G_1(\mathcal{A})] - \Pr[G_4(\mathcal{A})] \\ &= \sum_{i=1}^3 \Pr[G_i(\mathcal{A})] - \Pr[G_{i+1}(\mathcal{A})] \leq \text{Adv}_{\text{KD}}^{\text{prf}}(\mathcal{B}') + \text{Adv}_\Pi^{\text{ae}}(\mathcal{B}) + 2^{-\ell} . \end{aligned}$$

To justify Eq. (14), assume that the challenge bit c of game $\mathbf{G}_\Pi^{\text{ae}}(\mathcal{B})$ is 0. If $\mathcal{B}$ does not terminate prematurely then it perfectly simulates game $G_4(\mathcal{A})$. Thus the gap between the probabilities $\Pr[\mathbf{G}_\Pi^{\text{ae}}(\mathcal{B}) \Rightarrow \text{false} \mid c = 0]$ and $\Pr[G_4(\mathcal{A})]$ is at most the probability that $\mathcal{B}$ terminates prematurely, assuming that $c = 0$. View $\mathcal{B}$ as playing a balls-into-bins game. For each (partial) encrypted stream, we view it as throwing a ball to bin P^* , where P^* is the effective nonce prefix. Let

$d = \max\{4\ell, 4q/2n\}$. Our goal is to prove that the chance that some bin contains at least d balls is at most $2^{-\ell}$.

Note that $\mathcal{B}$'s queries are generated by $\mathcal{A}$, and the latter only receives truly random strings or $\perp$ that are independent of the effective nonce prefixes. In other words, $\mathcal{B}$'s queries are *non-adaptively* generated. For d balls that $\mathcal{B}$ throws, if there are two of them that belong to the same (i, R) then they can't go to the same bin. If none of them belongs to the same (i, R) then the d corresponding nonce prefixes are independent, uniformly random strings, and thus the chance that those balls go to the same bin is $2^{-(1-d)\ell}$. While Lemma 2 is stated for uniformly thrown balls, its proof only needs the hypothesis that for any d balls, the chance that they go to the same bin is at most $2^{-(1-d)\ell}$. Thus our goal above is a direct result of the extended Lemma 2.