COFFE: Ciphertext Output Feedback Faithful Encryption

On-Line Authenticated Encryption Without a Block Cipher

Christian Forler[1]*, David McGrew[2], Stefan Lucks[1], and Jakob Wenzel[1]

[1] Bauhaus-University Weimar, Germany, [2] Cisco Systems, USA
{Christian.Forler, Stefan.Lucks, Jakob.Wenzel}@uni-weimar.de
mcgrew@cisco.com

Abstract. In this paper we introduce the first authenticated encryption scheme based on a hash function, called **COFFE**. This research has been motivated by the challenge to fit secure cryptography into constrained devices – some of these devices have to use a hash function, anyway, and the challenge is to avoid the usage of an additional block cipher to provide authenticated encryption. **COFFE** satisfies the common security requirements regarding authenticated encryption, i.e., IND-CPA- and INT-CTXT-security. Beyond that, it provides the following additional security features: resistance against side-channel attacks and INT-CTXT security in the nonce-misuse scenario. It also support failure-friendly authentication under reasonable assumptions.

Keywords: authenticated encryption, provable security, side-channel , internet of things.

1 Introduction

The main goal for our work is to provide authenticated encryption in constrained implementation environments where communication security is required, such as devices connected to the Internet of Things. That class of devices typically has a small microprocessor with no direct hardware support for any cryptographic primitives, so that all cryptography must be implemented in software. There is only limited memory available to hold executable object code on these processors, so it is imperative to provide the needed cryptographic services in the most compact way possible. One way to achieve this compactness is through the careful implementation of cryptographic primitives. However, it is also possible to facilitate compactness for an overall system by minimizing the number of primitives that must be included in an implementation. In this work we present a design for an on-line authenticated encryption (AE) scheme suitable for restricted devices using a standardized or soon-to-be standardized hash function, e.g., SHA-1 [23], SHA-2 [24], or SHA-3 [4]. Implementations of this scheme can omit a block cipher mode of operation; this is a useful approach since the code size for the block cipher is typically greater than that of the hash function, and hash functions are used in public key cryptography as well.

We focus on the challenge of providing an authenticated encryption scheme that is easily accessible to developers. To provide this accessibility, we take the approach of defining a hash function mode of operation. That is, our AE scheme uses a cryptographic hash function as its only primitive, and does not require direct access to any hash function internals such as the compression function. We chose this approach based on feedback from the practice community. Hash function implementations are widely available, but these implementations do not provide interfaces to the compression function.

Note that to provide data privacy and data integrity, we transform the given hash function into a keyed hash function (PRF). On systems using restricted devices, due to the limited

* The research leading to these results received funding from the Sillicon Valley Community Foundation, under the Cisco Systems project*Misuse Resistant Authenticated Encryption for Complex and Low-End Systems (MIRACLE).*

Scheme	On-line	Side-Channel Res.	Misuse Res.	Rate-1
COFFE (this paper)	✓	✓	✓	✓
SpongeWrap [5]	✓	X	✓	✓
CHM [14]	✓	X	X	X
CIP[15]	✓	X	X	X
CWC [19]	✓	X	X	X
EAX [3]	✓	X	X	X
GCM [21]	✓	X	X	X
Generic Composition [2]	✓	X	X	X
HBS [16]	X	X	✓	X
SIV [26]	X	X	✓	X

Table 1. Comparison of existing AE-Schemes which can be instantiated with a PRF.

resources, it is desirable to minimize the cost of the code and the circuits for encrypting a message block [1], i.e., keep the size of the cryptographic footprint small. This means that we want only one costly operation per message block. We denote a scheme satisfying this property as a Rate-1 scheme. For example, the GCM authenticated encryption mode [21] is not a Rate-1 AE scheme, since it needs not only one block cipher call per message block, but also an additional galois field multiplication per message block, rendering GCM to be a Rate-2 AE scheme. Another example would be a Feistel-based scheme which requires at least three or four block cipher calls rendering such a scheme to a low-performance Rate-3 or Rate-4 scheme.

Since the implementation of an encryption scheme can be error prone (e.g., [6, 13, 18, 27, 28]) it would be desirable to provide a second line of defense to minimize the security fallout. A further preferable goal for our construction is to provide built-in resistance against side-channel attacks. Actually, the overlaying protocols, using an AE scheme, are responsible to provide this goal in an adequate form, e.g., TLS [8] and IPsec [12, 17] generate a new key for each session minimizing the measurements which can be done on the secret key. Obviously, an adversary can do a certain amount of measurements (depending on the size of the message) on the session key, but revealing the session key only compromises security for this specific encryption/decryption/authentication. Note that it does not compromise the currently used secret key. But, nevertheless, we provide side-channel resistance even if a protocol may fail to provide this kind of security.

We started our research by analyzing existing authenticated encryption schemes, where the block cipher within these schemes can be easily replaced by a keyed hash function. Unfortunately, none of those fulfill our requirements (see Table 1). As one can see, SpongeWrap [5] seems to be a very promising candidate, since it only lacks of built-in side-channel resistance. But, it belongs to the class of compression function based AE schemes, which yields to the fact that the internal used compression function can be seen as the real primitive to be used both for hashing and for authenticated encryption. This is basically not a technical problem, but, while cryptographers know what is meant by *the internal compression function*, typical standards, such as the SHA-2 standard [24], do not formally define it. So, without an explicit specification of a "new" cryptographic primitive, engineers (non-cryptographers) would not be likely to properly implement the authenticated encryption scheme. Also, while on many constrained devices "jumping" to the address of the internal compression function may be easy, this may be not the case for all such devices. In fact, we did consider this approach at the beginning of our research. It would even allow us to design a more efficient AE scheme

than the one we actually propose. But, due to the reasons discussed here, we made a decision against a purely compression function based AE scheme in favour of a hash function based AE scheme.

Contribution. In this paper we introduce **COFFE**, a novel hash function based on-line AE scheme which, for the best of our knowledge, is the first scheme which fulfills our stated requirements. It can be part of a minimal cryptographic suite that includes hashing and digital signatures. Because it is an authenticated encryption with associated data (AEAD) algorithm, it could be used in the AEAD interface of the Datagram TLS security protocol [25], which has been identified by the *IETF Constrained Application* working group as suitable for applications for internet of things. **COFFE** provides the standard INT-CTXT and IND-CPA (CCA3) security, one would expect from any good AE scheme, plus the following nonstandard security features.

1. *Misuse-resistant authenticity:* It is standard for an authenticated encryption scheme to claim and prove security against *nonce-respecting* adversaries. Almost always, security breaks apart if nonces are ever reused [11]. While the privacy of **COFFE** only holds for nonce-respecting adversaries (unlike the scheme presented in [11]), we prove that its authenticity does not depend on unique nonces.
2. *Failure-friendly authenticity:* The security proofs of most modes of operation assume PRF or PRP security for the underlying block cipher. If this fails, the mode is likely to be insecure. While the privacy of **COFFE** requires the hash function under a secret key to behave like a random function, authenticity can be established even for failing pseudo-randomness. Our mode ensures authenticity under a weaker unpredictability assumption assuming a strong key. Due to space limitation, we will only give a brief and informal discussion on this (cf. Section 4.1).
3. *Resistance against side-channel attacks:* Resistance against side-channel attacks is usually a matter of the implementation of a cryptosystem, rather than of the cryptosystem itself. Nevertheless, the design of a cryptosystem can contribute and ease side-channel resistant implementations. **COFFE** generates a new session key for each message depending on the nonce and the secret key, resulting in a side-channel resistant implementation in the nonce-respecting scenario.

Due to the upcoming event of the CAESAR competition[1], authenticated encryption has become a hot topic in the field of symmetric crypto. Nevertheless, since **COFFE** is designed to fill a quite particular nische we do not intent to submit it to the CAESAR competition.

Outline. In Section 2 we introduce our on-line authenticated encryption scheme **COFFE** and a possible practical instantiation using SHA-224. Section 3 contains essential preliminaries for the security proof. The security-analysis is given in Section 4. Section 5 concludes the paper.

2 COFFE

In this section we introduce **COFFE**, our novel on-line authenticated encryption scheme, which is inspired by the CFB and OFB modes of operation [9] following the idea of using the chaining value **and** the ciphertext as a feedback for the next iteration (see Figure 1). In contrast to other published AE schemes [3, 10, 14, 19, 21, 26], **COFFE** is based on a cryptographic hash function $F : \{0,1\}^* \to \{0,1\}^n$ instead of a block cipher. Furthermore, the

[1] `http://competitions.cr.yp.to/caesar.htm`

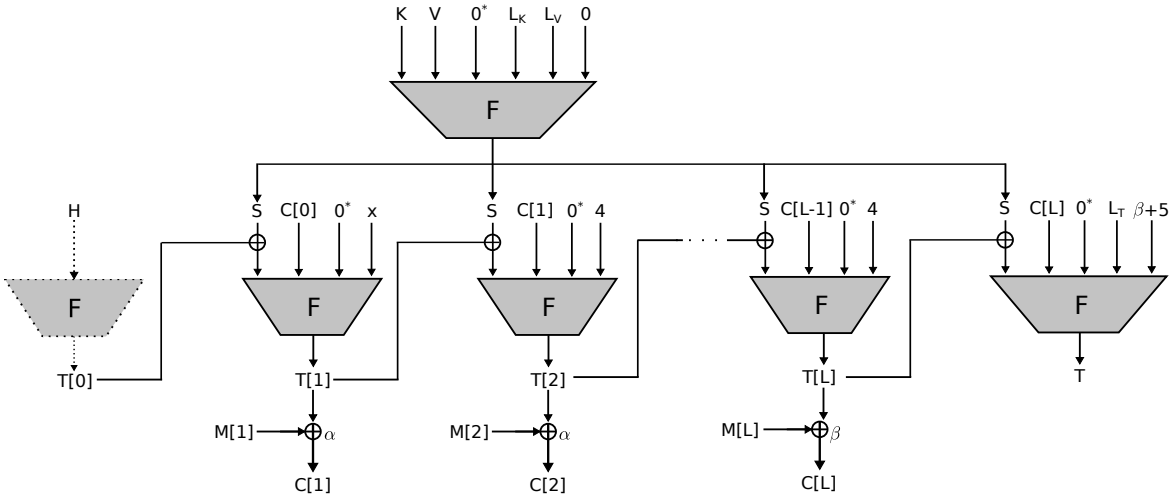

Fig. 1. Illustration of the encryption and authentication process of **COFFE**.

integrity of the ciphertext does not depend on a nonce, but only on the security of F. It is also the first AE scheme, designed to be resistant against side-channel attacks. Table 2 introduce the notions and variables used throughout this paper.

Identifier	**Description**
V	nonce (initial value)
H	header which is authenticated
M	plaintext
C	ciphertext
K	user-given secret key (long term key)
G	collision-resistant hash function processing the header
F, n	PRF-RK-secure one-way hash function with n bits of output
S	session key (short term key) generated from V and K
T_i	the i-th output of the hash function F
X_i	i-th block of a value X
L_x	length of x in bits
$X \mid\mid Y$	concatenation of two byte-strings X and Y
$X \oplus_i Y$	$\oplus$-operation of the i least significant bits of X and Y

Table 2. Notions used in the description of **COFFE**.

The definition of **COFFE** is given in Algorithm 1, where the left part of the algorithm denotes the function **EncryptAndAuthenticate** and the right part denotes the function **DecryptAndVerify**. As one can see, both functions consist of four steps, seemly separated by domains: session key generation (lines 10 and 20), processing of the header (lines 11 and 21), encryption/decryption (lines 12 and 22), and generation of the authentication tag (lines 13 and 23). The domain identifier is either encoded as a one- or two-byte value, depending on the choice of the hash function.

The term '0^*', which is used in each call to F, denotes a zero-padding, where the number of zeros depends on the input size of the used hash function (or more precisely the compression function) and the kind of padding which is used inside the hash function. Thus, it is always chosen that one needs only one compression function call for one hash function invocation, which complies with our Rate-1 design goal. Thus, we consider a constant m-bit input for F, producing an n-bit output with $m > n$.

Algorithm 1 COFFE

Input: V {Nonce}, H {Header}, M {Message}
Output: C {Ciphertext}, T {Tag}
10: $S \leftarrow F(K \mid\mid V \mid\mid 0^* \mid\mid L_K \mid\mid L_V \mid\mid \mathtt{0})$
11: $(x, T_0) \leftarrow$ **ProcessHeader**(H, F)
12: $(C, T_L) \leftarrow$ **ProcessMessage**(S, T_0, M, x)
13: $T \leftarrow F(T_L \oplus M_L \mid\mid C_L \mid\mid 0^* \mid\mid L_T \mid\mid L_{M_L} + 5)$
14: **return** (C, T)

Input: V {Nonce}, H {Header}, C {Ciphertext}
Output: M {Message}
20: $S \leftarrow F(K, V, 0^*, |K|, |V|, \mathtt{0})$
21: $(x, T_0) \leftarrow$ **ProcessHeader**(H, F)
22: $(M, T_L) \leftarrow$ **ProcessCiphertext**(S, T_0, C, x)
23: $T' \leftarrow F(T_L \oplus M_L \mid\mid C_L \mid\mid 0^* \mid\mid L_T \mid\mid L_{M_L} + 5)$
24: **if** $T \neq T'$ **then**
25: $\quad M \leftarrow \perp$
26: **end if**
27: **return** M

Step 1: Session Key Generation. Domain `0` is used to generate the session key S (*short term key*), which is derived from the secret key K (*long term key*) and the nonce V as shown in lines 10 and 20 of Algorithm 1. Note that L_K, L_V, and the domain description value are encoded as one- or two-byte values depending on the size of the key, and describe at least the three least significant bytes of the input. For practical applications we recommend to use a key size of n bits. Furthermore, the idea of the session key generation step is to support a built-in side-channel resistance due to the fact that a possible adversary can only evaluate one measurement on K per message. Moreover, due to the lack of a key scheduler in a hash function based setting, changing the key is free. **COFFE** benefits from this by using the secret key K only once for processing a message or a ciphertext. Therefore, in the nonce-respecting scenario, the number of measurements for K and S an adversary can make are limited.

Algorithm 2 ProcessHeader

Input: H {Header}, n {Output Length}
Output: (x, T_0) {Initialization Pair}
10: $L_H \leftarrow len(H)$
11: **if** $L_H < n$ **then**
12: $\quad$ **return** $(1, H \mid\mid 10^*)$
13: **else if** $L_H = n$ **then**
14: $\quad$ **return** $(2, H)$
15: **else**
16: $\quad$ **return** $(3, F(H))$
17: **end if**

Step 2: ProcessHeader. In this step we describe the processing of the associated data H (header), which can be of arbitrary length. Note that the domain x and the n-bit initial chaining value T_0 used in the processing of the first message block depend on the size of the associated data. These two values are computed as shown in Algorithm 2, leading to a

collision-resistant hash function. The goal of this definition of **ProcessHeader** is to achieve pair-wise distinct tuples (x, T_0) for pair-wise distinct values H and H'. Under the assumption that there is no collision, we have

$$H \neq H' \implies (x, T_0) = \textbf{ProcessHedaer}(H) \neq \textbf{ProcessHedaer}(H') = (x', T_0'),$$

not necessarily meaning that $x \neq x'$.

Remark. We use this separation process for security and performance reasons. Domain '1' is for the case that the authenticated data is less than one input block, thus, requiring 10^*-padding to generate a full block. Domain '2' is for associated data actually fitting into exactly one block, where no padding is applied. When the data is larger than one block, we must apply the hash function. This case is represented by Domain '3'. Omitting the invocation of the function F for small headers (i.e., $L_H \leq n$) increases significantly the performance when small messages with tiny headers are processed, e.g., IP packets.

Step 3: Encryption/Decryption. **COFFE** is generating a keystream which is XORed to a message to either encrypt or decrypt it. Since our scheme is designed to comply with the requirements of the use of standardized building block, it is construed to work with hash functions like SHA-1 and SHA-2. Thus, the input of the compression function is usually limited to less than $2n$ bits, due to the message padding. Note that the n-bit session key S and the domain separation value are mandatory inputs and hence, we have only less then n-bit left. To provide adequate security against forgery attacks, we need to additionally process two from the three following values: keystream block T_{i-1}, message block M_{i-1}, and ciphertext block C_{i-1}. More precisely, if we only use T_{i-1} in the next iteration step, the tag would become message-independent, i.e., the tag would not provide any integrity at all. Furthermore, if we use only C_{i-1} or M_{i-1}, omitting T_{i-1}, the tag value would only depend on the last ciphertext or plaintext block, respectively. Note that even processing only a single of those values is not possible using a naive approach without violating our requirement of one compression function call per hash function invocation. Thus, we decided to use the inputs to F in the following manner:

- n-bit value $S \oplus T_{i-1}$
- δ-bit domain separation value
- $(\alpha < n - \delta)$-bit ciphertext block C_{i-1}.

Our approach puts the hash function under a lot of stress, since it violates the PRF independency assumption. Thus, we now have to assume F to be PRF-secure in the XOR-related-key model. More precisely, an adversary has partial control over the key-input to F, resulting in a chance to produce a collision $S \oplus T_{i-1} = S' \oplus T_{j-1}$ for two distinct keys $S \neq S'$. Our security analysis in Section 4 shows that our approach still satisfies the birthday bound security.

Let $M = M_1, M_2, \ldots, M_L$ denote the message, where $L = \lceil L_M/\alpha \rceil$ is the number of message blocks processed. Here, all but the last blocks of M and C are of size α bits. The last blocks of M and C consist of at most α bit. Then, the encryption and decryption process of **COFFE** is defined in Algorithm 3, where $\textbf{ToHex}(\pi)$ (see lines 10 and 20) outputs the first $\alpha/4$ post decimal numbers of π interpreted as hex values ($C_0 = 0x1415926\ldots$).

Step 4: Tag Generation. In the final step we derive the authentication tag from the last chaining value T_L and the last ciphertext C_L as shown in lines 13 and 23 of Algorithm 1. Note that the length of the tag is constrained by the output size of F, e.g., at most n bits. The last domain allows a user to authenticate the header without any message to encrypt. Thus, the value β can become zero, but for F, $\beta + 5$ is always in the range $[5, \ldots, L_{M_L} + 5]$.

Algorithm 3 ProcessMessage/ProcessCiphertext

Input: S {Session Key}, T_0 {Initial Chaining Value}, M {Message}, x {Domain Specifier}	**Input:** S {Session Key}, T_0 {Initial Chaining Value}, C {Ciphertext}, x {Domain Specifier}
Output: C {Ciphertext}	**Output:** M {Message}
10: $C_0 \leftarrow \mathbf{ToHex}(\pi)$	20: $C_0 \leftarrow \mathbf{ToHex}(\pi)$
11: $T_1 = F(S \oplus T_0, C_0, 0^*, x)$	21: $T_1 = F(S \oplus T_0, C_0, 0^*, x)$
12: $C_1 = M_1 \oplus_\alpha T_1$	22: $M_1 = C_1 \oplus_\alpha T_1$
13: **for** $i = 2 \rightarrow L - 1$ **do**	23: **for** $i = 2 \rightarrow L - 1$ **do**
14: $\quad T_i = F(S \oplus T_{i-1}, C_{i-1}, 0^*, \mathtt{4})$	24: $\quad T_i = F(S \oplus T_{i-1}, M_{i-1}, 0^*, \mathtt{4})$
15: $\quad C_i = M_i \oplus_\alpha T_i$	25: $\quad M_i = C_i \oplus_\alpha T_i$
16: **end for**	26: **end for**
17: $C_L = M_L \oplus_\beta T_L$	27: $M_L = C_L \oplus_\beta T_L$
18: **return** C	28: **return** M

2.1 COFFE-SHA-224 – A Practical Instantiation

In this section we discuss a practical instantiation of **COFFE** using SHA-224 as the underlying hash function – called **COFFE-SHA-224**. First, we justify our usage of SHA-224 over SHA-256 as the underlying hash function.

Hash Function Choice. For the practical instantiation of **COFFE**, we where looking for a common, standardized, and flawless hash function which is suitable to be applied with restricted devices, where usually the size of a register is at most 32 bits. Thus, we made our choice in favour of a 32-bit optimized hash function, which renders SHA-224 and SHA-256 reasonable candidates to look at. Both SHA-224 and SHA-256 share the same compression function $f : \{0,1\}^{256} \times \{0,1\}^{512} \rightarrow \{0,1\}^{256}$. It compresses a 256-bit chaining value and a 512-bit message block into a 256-bit output value. These two hash function standards differ in two properties 1) they are using different initial values, and 2) SHA-224 truncates the output of the final compression function invocation while SHA-256 does not. Following the Merkle-Damgård paradigm [7, 22], SHA-224 and SHA-256 apply the secure 10^*-padding followed by a 64-bit value encoding the message length. Thus, the maximum possible input size to fit our requirements is given by $512 - 1 - 64 = 447$ bit. Due to the sake of simplification, we consider only byte-aligned values and we assume all values to be encoded octet-strings. Thus, we can only process message blocks with a size up to 440 bit, i.e., 55 byte. Using SHA-256 implies a 256-bit chaining value and thus, *only* 184 bits are left for the remaining input, including the domain separation byte and the previous ciphertext block. Furthermore, the tag generation step requires two additional input bytes – the length of the last message block β and the tag length L_T. Hence, we can process 160-bit message blocks. Since the size of the hash value of SHA-224 is reduced by 32 bits in comparison to SHA-256, we can process message blocks of 192 bits, which leads to an estimated performance speedup of about 20% in comparison to SHA-256. Furthermore, the 224-bit session key used in SHA-224 is sufficient to make practical attacks infeasible. This makes SHA-224 a logical choice for **COFFE**.

Parameter Choice. Here, we introduce a sound parameter choice for **COFFE-SHA-224** depending on the applied hash function SHA-224. The first step is to replace the function F from Algorithm 1 by SHA-224. This obviously leads to a size of 224 bits for the chaining values T_i. Based on our discussion above, we can process message blocks up to 192 bits ($\alpha = 192, \beta \leq 192$), i.e., we need only one byte to encode the domain specifier for the tag generation ($5 + \beta < 256$). One the one hand, the internal state of **COFFE** is larger than those of other common published AE schemes, which usually support a block size of 128 bits. On the other hand, **COFFE** employs a slightly worse ratio between the block size and the size of the internal state. Nevertheless, due to the larger block size, the performance

of **COFFE** is still reasonable, i.e., approximately 85% of SHA-224. To ensure an adequate security, we set the default parameter of the size of the secret key to 224 bits ($L_K = 224$) and the size of the nonce to 192 bits ($L_V = 192$).

3 Technical Preliminaries

Notions. Let $\{0,1\}^{n^*}$ denote an arbitrary number of blocks of size n and let $\perp$ denote a rejection of a message, i.e., that the verification of a message failed. Furthermore, we write 0^y for a string of y zero-bits, and 0^* for a string of zero or more zero-bits to fill a given input for F up to exactly the required input size.

We define an adversary as a computationally unbounded but always-halting algorithm A. In this paper we assume wlog. that the adversary A never asks a query which answer is already known. Further, we denote A^O for an adversary A with access to an oracle O.

Keyed Hash Function. A keyed hash function $F : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^n$ is a family of functions that computes a fixed-size hash value $Y \in \{0,1\}^n$ from a message $X \in \{0,1\}^*$ of arbitrary length under a given key $K \in \{0,1\}^k$. We write $Y = F_K(X)$ for $Y = F(K, X)$.

It is easy to build a keyed hash function from an un-keyed hash function by dividing the input space into two parts, the key space and a new message space. In our approach, the key is derived by the session key S and the previous chaining value T_{i-1}. Note, following Liskov et al. [20], these two values can be seen as a tweak and hence, we name this type of functions as *tweakable keyed hash functions*.

Since we combine S and T_{i-1} using an XOR operation, the underlying primitive of **COFFE** must be secure in the XOR-related-key model. Thus, we define the XOR-related-key PRF (PRF-XRK) security of a keyed hash function F by the success probability of an adversary trying to differentiate between the keyed hash function and a random function. In this scenario the adversary can freely choose the *tweak* input. In the following, we define three security notions, which are necessary to proof the security of our scheme.

Definition 1 (PRF). *Let $F : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^n$ be a keyed hash function with a secret key $K \xleftarrow{\$} \mathcal{K}$, and A a fixed adversary. Let $\$(\cdot)$ denote a random bit oracle which returns always n-bit random values. The* PRF *advantage of A in distinguishing F from a random function is defined as*

$$\mathbf{Adv}_F^{\mathrm{PRF}}(A) = \left| \Pr_K \left[A^{F_K(\cdot)} \Rightarrow 1 \right] - \Pr \left[A^{\$(\cdot)} \Rightarrow 1 \right] \right|.$$

The PRF *advantage among all adversaries that run in time at most t and make at most q queries to the available oracle is given by*

$$\mathbf{Adv}_F^{\mathrm{PRF}}(q,t) = \max_A \{ \mathbf{Adv}_F^{\mathrm{PRF}}(A) \}.$$

Definition 2 (PRF-XRK). *Let $F : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^n$ be a keyed hash function with a secret key $K \xleftarrow{\$} \mathcal{K}$, and A a fixed adversary. Let $\$(\cdot)$ denote a random bit oracle which returns always n-bit random values. The* PRF-XRK *advantage of A in distinguishing F from a random function is defined as*

$$\mathbf{Adv}_F^{\text{PRF-XRK}}(A) = \left| \Pr_K \left[A^{F_{\oplus(K,\cdot)}(\cdot)} \Rightarrow 1 \right] - \Pr \left[A^{\$(\cdot)} \Rightarrow 1 \right] \right|.$$

The PRF-RK *advantage among all adversaries that run in time at most t and make at most q queries to the available oracle is given by*

$$\mathbf{Adv}_F^{\text{PRF-XRK}}(q,t) = \max_A \left\{ \mathbf{Adv}_F^{\text{PRF-XRK}}(A) \right\}.$$

Note that this is a stronger assumption than the PRF security model as you can easily reduce PRF-XRK security to PRF security by fixing the tweak input. Thus, we have

$$\mathbf{Adv}_F^{\text{PRF}}(q,t) \leq \mathbf{Adv}_F^{\text{PRF-XRK}}(q,t).$$

Authenticated Encryption (With Associated Data). An authenticated encryption scheme is a triple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. It aims to provide both privacy and data integrity. The key generation function $\mathcal{K}$ takes no input and returns a randomly chosen key K from the key space $\{0,1\}^k$. The encryption algorithm $\mathcal{E}$ and the decryption algorithm $\mathcal{D}$ are deterministic algorithms that map values from $\{0,1\}^k \times \{0,1\}^{n^*} \times \{0,1\}^v \times \{0,1\}^{n^*}$ to a byte string or – if the input is invalid – the value $\perp$. For sake of convenience, we usually write $\mathcal{E}_K(H,V,M)$ for $\mathcal{E}(K,H,V,M)$ and $\mathcal{D}_K(H,V,M)$ for $\mathcal{D}(K,H,V,M)$, where the message M and the associated data H are chosen from the set $\{0,1\}^{n^*}$, a key K from the key space $\{0,1\}^k$, and a nonce V from the nonce space $\{0,1\}^v$. We require $\mathcal{D}_K(H,V,\mathcal{E}_K(H,V,M)) = M$ for any possible quadruple (K,V,H,M).

Definition 3 (CCA3). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an authenticated encryption scheme, A a fixed adversary, and $K \xleftarrow{\$} \mathcal{K}$ be a randomly chosen key. The* CCA3 *advantage of A in breaking Π is defined as*

$$\mathbf{Adv}_\Pi^{\text{CCA3}}(A) = \left| \Pr_K \left[A^{\mathcal{E}_K(\cdot,\cdot,\cdot),\mathcal{D}_K(\cdot,\cdot,\cdot)} \Rightarrow 1 \right] - \Pr \left[A^{\$(\cdot,\cdot,\cdot),\perp(\cdot,\cdot,\cdot)} \Rightarrow 1 \right] \right|. \tag{1}$$

The adversary's random-bit oracle $\$(\cdot,\cdot,\cdot)$ returns on a plaintext query $(H,V,M) \in \{0,1\}^{n^*} \times \{0,1\}^v \times \{0,1\}^{n^*}$ a random string of length $|\mathcal{E}_K(H,V,M)|$. The $\perp(\cdot,\cdot,\cdot)$ oracle returns $\perp$ on every input. In this paper we assume that an adversary never asks a query for which the answer is already known, e.g., if it has received the result for an encryption query $(C,T) = \mathcal{E}(V,H,M)$, it will never ask for $(V,H,M) = \mathcal{D}(C,T)$, and vice versa. Bellare and Namprempre have shown in [2] that one can rewrite Equation 1 as

$$\mathbf{Adv}_\Pi^{\text{CCA3}}(q,\ell,t) \leq \mathbf{Adv}_\Pi^{\text{IND-CPA}}(q,\ell,t) + \mathbf{Adv}_\Pi^{\text{INT-CTXT}}(q,\ell,t), \tag{2}$$

where $\mathbf{Adv}_\Pi^{\text{CCA3}}(q,\ell,t)$ is the maximum advantage of all CCA3-adversaries, which run in time at most t, asks a total maximum of q queries to $\mathcal{E}$ and $\mathcal{D}$, and whose total query length is at most ℓ blocks. Then, the IND-CPA-advantage is given by

$$\mathbf{Adv}_\Pi^{\text{IND-CPA}}(q,\ell,t) \leq \left| \Pr_K \left[A^{\mathcal{E}_K(\cdot,\cdot,\cdot)} \Rightarrow 1 \right] - \Pr \left[A^{\$(\cdot,\cdot,\cdot)} \Rightarrow 1 \right] \right|.$$

Furthermore, the advantage of an INT-CTXT-adversary is given by the success probability of winning the game $G_{\text{INT-CTXT}}$ defined in Figure 2. Thus, we have

$$\mathbf{Adv}_\Pi^{\text{INT-CTXT}}(A) \leq \Pr[A^{G_{\text{INT-CTXT}}} \Rightarrow 1],$$

where

$$\mathbf{Adv}_\Pi^{\text{INT-CTXT}}(q,\ell,t) \leq \max_A \left\{ \mathbf{Adv}_\Pi^{\text{INT-CTXT}}(A) \right\}.$$

For convenience, we introduce a notation for a *restriction on a set*. Let $\mathcal{Q} = \mathcal{A} \times \mathcal{B} \times \mathcal{C}$, then we denote $\mathcal{Q}_{|B,C} = \{(B,C) \mid \exists A : (A,B,C) \in \mathcal{Q}\}$ as the restriction of $\mathcal{Q}$ to B and C with $A \in \mathcal{A}, B \in \mathcal{B}$, and $C \in \mathcal{C}$. This generalizes in the obvious way.

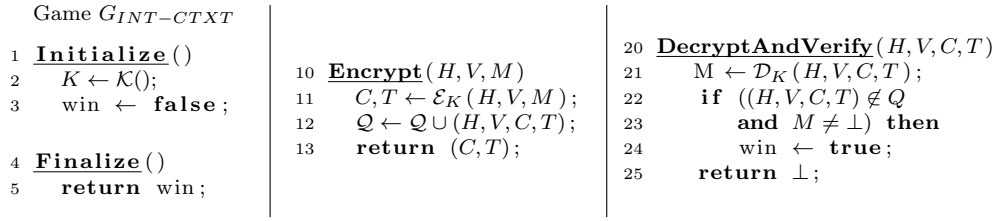

Fig. 2. Game $G_{\text{INT-CTXT}}$ is the INT-CTXT$_{\Pi}$ game where $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$.

4 Security

This section describes the security for our generic **COFFE** construction considered under the reasonable assumption that the size of the secret key K can be larger or equal to the size of the session key S, i.e., $|K| \geq |S|$. The first step is to show the CPA-security when considering a nonce-respecting adversary. For the INT-CTXT-proof, we generalize the adversary by allowing it to reuse a nonce, i.e., transforming it to a nonce-ignoring adversary.

Theorem 1. *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a **COFFE** scheme as in Algorithm 1, i.e., $\mathcal{K}$ is the key derivation function, $\mathcal{E} =$ **EncryptAndAuthenticate** and $\mathcal{D} =$ **DecryptAndVerify**. Then,*

$$\begin{aligned}\mathbf{Adv}^{\text{CCA3}}_{\Pi}(q, \ell, t) \;&\leq\; \frac{8\ell^2 + 3q^2}{2^n} + 2 \cdot \mathbf{Adv}^{\text{PRF-XRK}}_{F_*}(q, \ell, t) \\ &\quad + \frac{3\ell^2 + 2q^2}{2^n} + \frac{q}{2^{L_T}} + 2 \cdot \mathbf{Adv}^{\text{PRF-XRK}}_{F_*}(q + \ell, O(t)) \\ &\leq\; \frac{11\ell^2 + 5q^2}{2^n} + \frac{q}{2^{L_T}} + 4 \cdot \mathbf{Adv}^{\text{PRF-XRK}}_{F_*}(q, \ell, t),\end{aligned}$$

where L_T denotes the length of the tag value in bits.

Proof. The proof follows from Equation 2 together with Lemma 1 and Lemma 2. □

Lemma 1. *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a **COFFE** scheme as in Algorithm 1. Let q be the number of total queries an adversary A is allowed to ask and ℓ be an integer representing the total length in blocks of the queries to $\mathcal{E}$. Then,*

$$\begin{aligned}\mathbf{Adv}^{CPA}_{\Pi}(q, \ell, t) &\leq \frac{2(\ell + q)^2 + 2\ell^2 + q^2}{2^n} + 2 \cdot \mathbf{Adv}^{\text{PRF-XRK}}_{F_*}(q, \ell, t) \\ &\leq \frac{8\ell^2 + 3q^2}{2^n} + 2 \cdot \mathbf{Adv}^{\text{PRF-XRK}}_{F_*}(q, \ell, t).\end{aligned}$$

Proof (Lemma 1). This proof is using common game playing arguments. As stated above, the length of the secret key K can differ from the length of the session key S. If this is the case, we can partition F into two independent PRF's: $F_1 : \{0,1\}^{|K|} \times \{0,1\}^* \to \{0,1\}^n$ for the generation of the session key and $F_2 : \{0,1\}^{|S|} \times \{0,1\}^* \to \{0,1\}^n$ for the processing of the message and the generation of the authentication tag. Due to the domain separation, the partitioning of F is still valid if $|K| = |S|$, i.e., the domain of the session key generation is always 0 and the domain of the message processing is always > 0.

Under this assumption, we can replace the functions F_1 by a PRF. This can be upper bounded by

$$\mathbf{Adv}^{\text{PRF}}_{F_1}(q, O(t)).$$

Furthermore, we can replace the function F_2 by a PRF-XRK, since the adversary has partial control over the key $S \oplus T[i]$. This can be upper bounded by

$$\mathbf{Adv}_{F_2}^{\text{PRF-XRK}}(q+\ell, O(t)).$$

Due to the sake of simplification, we define

$$\mathbf{Adv}_{F_*}^{\text{PRF-XRK}}(q,\ell,t) = \max\left\{\mathbf{Adv}_{F_2}^{\text{PRF-XRK}}(q+\ell, O(t)), \mathbf{Adv}_{F_1}^{\text{PRF}}(q, O(t))\right\}.$$

In the following analysis we always consider the full output length n of the tag generation step, i.e., even if L_T is smaller than n, we skip the truncation step for the proof. This is valid, since showing CPA-security for the tag generation step without truncation implies CPA-security for the tag generation with truncation. Assume an adversary B which is successful by attacking the truncated version, then we can build an adversary B' using B to be successful with the same probability attacking the untruncated version.

Next, we denote $\mathcal{Q}$ as the query history of the adversary, where $\mathcal{Q}_{|T_\mu^i, T^i}$ contains the produced keystream T_μ^i with $1 \leq \mu \leq \ell$ and the tags T^i with $1 \leq i \leq q$. Note that all samples T_μ^i and T^i are output values of the hash function F_2. We can say that **COFFE** is CPA-secure, if the produced keystream and the tag values within the query history are indistinguishable from a sequence R of distinct random values of the same size, where the length of this sequence is limited to $\ell + q$. It is easy to see that this event can be upper bounded by

$$\frac{(\ell+q)^2}{2^n}.$$

To complete our proof, we have to estimate the probability $\Pr[Dist]$ that all values within the list $\mathcal{Q}_{|T_\mu^i, T^i}$ are distinct. Therefore, we upper bound the probability $\Pr[Coll]$ for a collision of at least two of the values within this list, since

$$\Pr[Dist] = 1 - \Pr[Coll].$$

To upper bound $\Pr[Coll]$, we first consider the input parameter of F_2 represented by the quadruple (S^i, T_μ, C_μ, d). Note that we ignore the 0^*-padding, which leads to a higher success probability for an adversary. Let $Z_i = (S^i, T_\mu^i, C_\mu^i, d_i)$ and $Z_j = (S^j, T_\nu^j, C_\nu^j, d_j)$ with $1 \leq i, j \leq q$ and $1 \leq \mu, \nu \leq L^*$, where L^* denotes the number of blocks of the longest message. A collision between two such tuples is given either when we have found a collision for F or we have found an input collision for the values $S^i \oplus T_\mu^i = S^j \oplus T_\nu^j$. For our case analysis (cf. Table 3), we encode the difference between two such input tuples Z_i and Z_j using a five-bit value. For example, the value "10110" is defined as follows:

$$10110 := \begin{cases} i \neq j \\ T_\mu^i = T_\nu^j \\ S^i \oplus T_\mu^i \neq S^j \oplus T_\nu^j \\ C_\mu^i \neq C_\nu^j \\ d_i = d_j, \end{cases}$$

where $1 \leq i, j \leq q$ and $1 \leq \mu, \nu \leq L^*$.

Note that Table 3 contains a complete case analysis, since all possible cases are covered. The cases which occur with a zero-probability are obviously seen as impossible and marked by "–". The reason for the occurrence of these cases is a violation of the XOR relation between the values S^i and T_μ^i or S^j and T_ν^j, respectively. For example, $S^i = S^j, T_\mu^i = T_\nu^j$, and $S^i \oplus T_\mu^i \neq S^j \oplus T_\nu^j$ is an impossible case. The Case "00000" implies that a collision must

Case	Event	Case	Event	Case	Event	Case	Event
00000	trivial	01000	–	10000	1	11000	2,4
00001	3	01001	–	10001	1	11001	3
00010	3	01010	–	10010	1	11010	3
00011	3	01011	–	10011	1	11011	3
00100	–	01100	3	10100	3	11100	1,3
00101	–	01101	3	10101	3	11101	3
00110	–	01110	3	10110	3	11110	3
00111	–	01111	3	10111	3	11111	3

Fig. 3. This table illustrates the case analysis for the proof of Lemma 1, where each case with a non-zero probability is covered by at least one event. The case "11000" is covered by two events depending on the considered domain (Event 2 covers the domain 1,2, and 3; Event 4 covers all other domains). The second special case "11100" is covered by Event 1 if $S^i = S^j$ and by Event 3 if $S^i \neq S^j$.

have happened before in the same query and is already covered by one of the other non-zero cases. In the following we analyze four events which cover all remaining cases with a non-zero probability given in Table 3.

After asking at most q queries, we check the adversaries query history $\mathcal{Q}$ – which contains all queries and their results – for the occurrence of bad events. We let the adversary win immediately if one of the bad events becomes true. Let denote A_y the y-th event and A_z the z-th event. The occurrence of an event A_y implies that no event A_z with $z \in \{1, \ldots, y-1\}$ occurred before. Hence, the order of the events matters.

Event 1: Session Key Collision. The first case describes the scenario where an adversary finds two values S^i and S^j, generated using the function F_1, with $i \neq j$, $S^i = S^j$, and $1 \leq i, j \leq q$. This can be upper bounded by

$$q^2/2^n.$$

Since F_1 and F_2 are independent, all values S^i are independent from the values T^i (tag values) and T^i_μ (chaining values) with $1 \leq \mu \leq L^*$.

If an adversary is able to find two values S^i and S^j with $i \neq j$ and $S^i = S^j$, then it is able to distinguish **COFFE** from a PRF using the following attack with a complexity of $O(1)$. Consider two queries (H, V, M) and (H, V', M'), where $M \neq M'$ are two single-block messages of the same length, i.e., $L_M = L_{M'}$ and $V \neq V'$ with $S = S'$. Then, it is obvious that $T_0 = T'_0$ and $C_0 = C'_0$. For the distinguishing attack we test whether $M_1 \oplus C_1 = M'_1 \oplus C'_1$. Therefore, we let the adversary win if it founds two nonces $V \neq V'$ which lead to the same $S = S'$.

Event 2: Input Collision – Associated Data. In this case we consider an adversary which finds two pairs $(T^i_0 \oplus S^i, x^i)$ and $(T^j_0 \oplus S^j, x^j)$ with $T^i_0 \oplus S^i = T^j_0 \oplus S^j$, $x^i = x^j$, and $i \neq j$. This leads to two colliding inputs for F in the first iteration. If no collision occurs, all T^i_1 are independent random values. The success probability for this case can be upper bounded by

$$\frac{(q+\ell)^2}{2^n}.$$

Event 3: Output Collision. For this case we consider an adversary which finds two values T^i_μ and T^j_ν with $T^i_\mu = T^j_\nu$, $1 \leq \mu, \nu \leq L^*$, $1 \leq i, j \leq q$, and $(\mu, i) \neq (\nu, j)$. This can be upper bounded by

$$\ell^2/2^n.$$

If no collision is found, the values T_L^i and T_L^j with $i \neq j$ must differ, where L denotes the index of the last chaining value. This implies that all authentication tags T^i can be seen as independent random values.

Event 4: Input Collision – Message and Tag. In this case we consider an adversary which finds two values $T_\mu^i \oplus S^i$ and $T_\nu^j \oplus S^j$ with $T_\mu^i \oplus S^i = T_\nu^j \oplus S^j$ for $1 < \mu, \nu \leq L^*$ and $1 \leq i, j \leq q$. This leads to two colliding inputs for F_2. Note that we assume that the adversary did not find an output collision before. The probability for this event can be upper bounded by

$$\ell^2/2^n.$$

Our claim follows by adding up the individual bounds. □

Lemma 2. *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a **COFFE** scheme as in Algorithm 1. We assume the adversary to be nonce-ignoring, i.e., it is able to choose two nonces $V_i = V_j$ with $i \neq j$. Then,*

$$\mathbf{Adv}_{\Pi}^{\text{INT-CTXT}}(q, \ell, t) \leq \frac{3\ell^2 + 2q^2}{2^n} + \frac{q}{2^{L_T}} + 2 \cdot \mathbf{Adv}_{F_*}^{\text{PRF-XRK}}(q + \ell, O(t)),$$

where L_T denotes the length of the tag value in bits.

The proof of Lemma 2 is given in Appendix A.

4.1 Authenticity under Weaker Assumptions (Sketch)

In the following we analyze the case where the underlying primitive of **COFFE** is not PRF-RK-secure. For example, assume that $c \geq 1$ bits of the hash function output T_i are constant. Then, privacy is gone for good, since the adversary can easily learn c bits of the message block M_i. But, as it turns out, the core structure of **COFFE** still preserves message integrity under a reasonable "unpredictability" assumption, since an INT-CTXT-adversary apparently must *predict* the output of the hash function, not just distinguish it from random. This observation holds even for nonce-ignoring adversaries. We make the following assumption.

If S is a random variable, or hard to distinguish from one, we could just assume the outputs $F_S(\cdot, \cdot)$ to be hard to predict. But, if we do not assume F to be a good PRF, we cannot assume $S = F_K(\cdot, \cdot, \cdot)$ to be uniformly and randomly distributed. We solve this by defining a function

$$F_K^2(X, Y, Z) := F(F(K \oplus X, Y) \oplus K, F(K \oplus X, Y) \oplus Z).$$

It is straightforward to rewrite **COFFE** as the sequential application of $F_K^2(\cdot, \cdot, \cdot)$, rather than as the sequential application of $F_S(X, Y) := F(S \oplus X, Y)$ with $S = F_K(V, L_K, L_V)$. Note that we do not pay attention to the 0^*-padding and the domain. This leads to a higher success probability for the adversary.

Unpredictability of F^2: Fix the key length L_K and the length L_T of the authentication tag. Choose a secret L_K-byte key K. Allow the adversary to make queries to the oracle F_K^2. Let denote $\mathcal{Q}$ the query history containing all queries (X_i, Y_i, Z_i), with $1 \leq i \leq q$, made by the adversary. We assume that no efficient adversary can find any tuple (X, Y, Z, T) with $(X, Y, Z) \notin \mathcal{Q}$ and $F_K^2(X, Y, Z) = T$.

Under this assumption, forgeries (INT-CTXT-attacks) are infeasible. For **COFFE**, a forgery is a triple (nonce, ciphertext, tag) that is neither the output from the encryption oracle nor rejected by the decryption oracle – the only event, that would allow an INT-CTXT-adversary to win its game. *Wlog.* we assume the (nonce, ciphertext) pair of a forgery has not been used as the output of an encryption query, before. Thus, there has either been a *weak collision*, i.e., two calls of F^2 under the secret key K with different inputs share the same output, or, there has been no such collision, and the INT-CTXT-adversary has correctly predicted the tag – directly violating our assumption. But even a weak collision is a violation of the unpredictability assumption, since an adversary finding a weak collision within q queries can easily be turned into an adversary predicting $F_K^2(\cdot,\cdot,\cdot)$ with probability $> 2/q^2$. (For an analysis aiming at good concrete security, we might formally assume weak collision resistance for F^2.)

Cryptographic schemes could be used out of the specifications' range or employing a primitive not quite as strong as demanded. Ideally, such schemes should provide a *second line of defense.* The brief analysis of **COFFE** under an unpredictability assumption, as well as our analysis regarding nonce-ignoring adversaries above, show that **COFFE** actually provides a second line of defense, *maintaining authenticity in situations where privacy could not be defended any more.*

5 Conclusion

In this paper we presented **COFFE**, the first provably-secure authenticated encryption scheme that has been natively designed for the usage of a hash function, rather than a block cipher as the underlying primitive. **COFFE** provides the security one would expect from a traditional scheme, plus three additional properties:

1. It provides reasonable resistance against side-channel attacks based on statistical information, since for each encryption process a new *short term key* is derived from a nonce and the *long term key.*
2. It provides ciphertext-integrity in a nonce-misuse scenario.
3. It provides ciphertext-integrity even if the underlying primitive leaks information about its input. This can be formally proven, based on some *unpredictability* and strong key assumptions.

Considering these properties, **COFFE** is designed to be well-suited for critical security applications in resource-restricted, embedded devices, which require strong security requirements by providing only a small cryptographic suite.

References

1. Jari Arkko, Carsten Bormann, Peter Friess, Cullen Jennings, Antonio Skarmeta, Zack Shelby, and Hannes Tschofenig. Report from the Smart Object Security Workshop. In *Smart Object Security Workshop*, 2012.
2. Mihir Bellare and Chanathip Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. *J. Cryptology*, 21(4):469–491, 2008.
3. Mihir Bellare, Phillip Rogaway, and David Wagner. The EAX Mode of Operation. In *FSE*, pages 389–407, 2004.
4. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The Keccak SHA-3 submission. Submission to NIST (Round 3), 2011.
5. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In *Selected Areas in Cryptography*, pages 320–337, 2011.

6. Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting Mobile Communications: The Insecurity of 802.11. In *MOBICOM*, pages 180–189, 2001.
7. Ivan Damgård. A Design Principle for Hash Functions. In *CRYPTO*, pages 416–427, 1989.
8. T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.
9. Morris Dworkin. *Special Publication 800-38A: Recommendation for block cipher modes of operation.* National Institute of Standards, U.S. Department of Commerce, December 2001.
10. Morris Dworkin. *Special Publication 800-38C: Recommendation for block cipher modes of operation: the CCM mode for authentication and confidentiality.* National Institute of Standards and Technology, U.S. Department of Commerce, May 2005.
11. Ewan Fleischmann, Christian Forler, and Stefan Lucks. Mcoe: A family of almost foolproof on-line authenticated encryption schemes. In *FSE*, pages 196–215, 2012.
12. D. Harkins and D. Carrel. The Internet Key Exchange (IKE). RFC 2409 (Proposed Standard), November 1998. Obsoleted by RFC 4306, updated by RFC 4109.
13. George Hotz. Console Hacking 2010 - PS3 Epic Fail. 27th Chaos Communications Congress, 2010. `http://events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf`.
14. Tetsu Iwata. New Blockcipher Modes of Operation with Beyond the Birthday Bound Security. In *FSE*, pages 310–327, 2006.
15. Tetsu Iwata. Authenticated Encryption Mode for Beyond the Birthday Bound Security. In Serge Vaudenay, editor, *Progress in Cryptology - AFRICACRYPT 2008, First International Conference on Cryptology in Africa, Casablanca, Morocco, June 11-14, 2008. Proceedings*, volume 5023 of *Lecture Notes in Computer Science*, pages 125–142. Springer, 2008.
16. Tetsu Iwata and Kan Yasuda. HBS: A Single-Key Mode of Operation for Deterministic Authenticated Encryption. In *FSE*, pages 394–415, 2009.
17. S. Kent. IP Encapsulating Security Payload (ESP). RFC 4303 (Proposed Standard), December 2005.
18. Tadayoshi Kohno. Attacking and Repairing the WinZip Encryption Scheme. In *ACM Conference on Computer and Communications Security*, pages 72–81, 2004.
19. Tadayoshi Kohno, John Viega, and Doug Whiting. CWC: A High-Performance Conventional Authenticated Encryption Mode. In *FSE*, pages 408–426, 2004.
20. Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable Block Ciphers. *J. Cryptology*, 24(3):588–613, 2011.
21. David A. McGrew and John Viega. The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In *INDOCRYPT, volume 3348 of LNCS*, pages 343–355. Springer, 2004.
22. Ralph C. Merkle. One Way Hash Functions and DES. In *CRYPTO*, pages 428–446, 1989.
23. NIST National Institute of Standards and Technology. FIPS 180-1: Secure Hash Standard. April 1995. See http://csrc.nist.gov.
24. NIST National Institute of Standards and Technology. FIPS 180-2: Secure Hash Standard. April 1995. See http://csrc.nist.gov.
25. E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard), jan 2012. `http://www.ietf.org/rfc/rfc6347.txt`.
26. Phillip Rogaway and Thomas Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In *EUROCRYPT*, pages 373–390, 2006.
27. Todd Sabin. Vulnerability in Windows NT's SYSKEY encryption. *BindView Security Advisory*, 1999. Availalbe at `http://marc.info/?l=ntbugtraq&m=94537191024690&w=4`.
28. Hongjun Wu. The Misuse of RC4 in Microsoft Word and Excel. Cryptology ePrint Archive, Report 2005/007, 2005. `http://eprint.iacr.org/`.

A Proof of Lemma 2

A.1 Proof Preliminaries

Length of Longest Common Prefix ($\mathbf{LLCP}_n$). The bit length of a string $x \in \{0,1\}^n$ is denoted by $|x| := n$. For integers $n, \ell, d \geq 1$, set $D_n^d = (\{0,1\}^n)^d$, $D_n^* := \bigcup_{d \geq 0} D_n^d$, and $D_{\ell,n} = \bigcup_{0 \leq d \leq \ell} D_n^d$. Note that D_n^0 only contains the empty string. For $M \in D_n^d$, we write $M = M_1, \ldots, M_d$ with $M_i \in D_n$ for $1 \leq i \leq d$. For $P, R \in D_n^*$, say, $P \in D_n^p$ and $R \in D_n^r$, we define the *length of the longest common n-prefix* of P and R as

$$\mathrm{LLCP}_n(P,R) = \max_i \{P_1 = R_1, \ldots, P_i = R_i\}\,.$$

For a non-empty set $\mathcal{Q}$ of strings in D_n^*, we define $\mathrm{LLCP}_n(\mathcal{Q}, P)$ as $\max_{q \in \mathcal{Q}}\{\mathrm{LLCP}_n(q, P)\}$. For example, if $P \in \mathcal{Q}$, then $\mathrm{LLCP}_n(\mathcal{Q}, P) = |P|/n$.

```
1  LLCP'(R, (H, V, M))
2  p ← 0;
3  for each (H', V', M') ∈ R do
4       if (H' = H and V' = V) then
5          p = max{p, LLCP_a(M', M)};
6  return p;
```

Fig. 4. LLCP' computes the LLCP_a of two inputs, a triple (H, V, M) and a set of tuples R, where a is the size of a message block.

A.2 Proof

```
1  Initialize()
2    K ←$ K();
3    B0, B1, B2, B3, B4, B5 ← ∅;
4    win ← false;

100 Encrypt(H, V, M) Game G1
101   L ← ⌈|M|/a⌉;
102   S ← F1(K || V || 0* || L_K || L_V || 0);
103   x, T0 ← G(H, F);
104   C0 ← κ(π);
105   I ← S ⊕ T0;
106   T1 ← F2(I || C0 || 0* || x);
107   C1 ← T1 ⊕α M1;
108   for i = 2, ..., L − 1 do
109     I ← S ⊕ T_{i−1};
110     T_i ← F2(I || C_{i−1} || 0* || 4);
111     C_i ← T_i ⊕α M_i;
112   b ← |M_L|;
113   I ← S ⊕ T_{L−1};
114   T_L ← F2(I || C_{L−1} || 0* || 4);
115   C_L ← T_L ⊕β M_L;
116   I ← S ⊕ T_L;
117   T ← F2(I || C_L || 0* || L_T || b + 5);
118   Q ← (H, V, C, T);
119   return (C1, ..., C_L, T);

5  Finalize()
6    return win;

119 DecryptAndVerify(H, V, C, T) Game G1
120   L ← ⌈|C|/a⌉;
121   S ← F1(K || V || 0* || L_K || L_V || 0);
122   x, T0 ← G(H, F);
123   C0 ← κ(π);
124   I ← S ⊕ T0;
125   T1 ← F2(I || C0 || 0* || x);
126   M1 ← T1 ⊕α C1;
127   for i = 2, ..., L − 1 do
128     I ← S ⊕ T_{i−1};
129     T_i ← F2(I || C_{i−1} || 0* || 4);
130     M_i ← T_i ⊕α C_i;
131   b ← |C_L|;
132   I ← S ⊕ T_{L−1};
133   T_L ← F2(I || C_{L−1} || 0* || 4);
134   M_L ← T_L ⊕β C_L;
135   I ← S ⊕ T_L;
136   T' ← F2(I || C_L || 0* || L_T || b + 5);
137   if (T =_{L_T} T') then
138     win ← true;
139   return ⊥;
```

Fig. 5. Games G_1 for the proof of Lemma 2. The variable a denotes the block size of the message and ciphertext blocks with $a \leq n$, where n is the output size of F, F_1, and F_2, each. The function $\kappa(\pi)$ returns the first $a/4$ post decimal positions of π interpreted as a string of hex characters, and L_T denotes the bit-length of the tag.

This proof borrows ideas from the INT-CTXT-proof presented by Fleischmann et al. [11].

Our bound is derived by game-playing arguments. Consider games G_1-G_3 of Figure 5 and Figure 6, and a fixed adversary A asking at most q queries with a total length of at most ℓ blocks. We assume that the adversary never asks for a query for which the answer is already known. The functions **Initialize** and **Finalize** are identical for all games in this proof. Lets denote G_0 as the INT-CTXT-game defined in Figure 2 (cf. Section 3). Therefore, we have

$$\mathbf{Adv}_{\Pi}^{\text{INT-CTXT}}(A) \leq \Pr[A^{G_0} \Rightarrow 1].$$

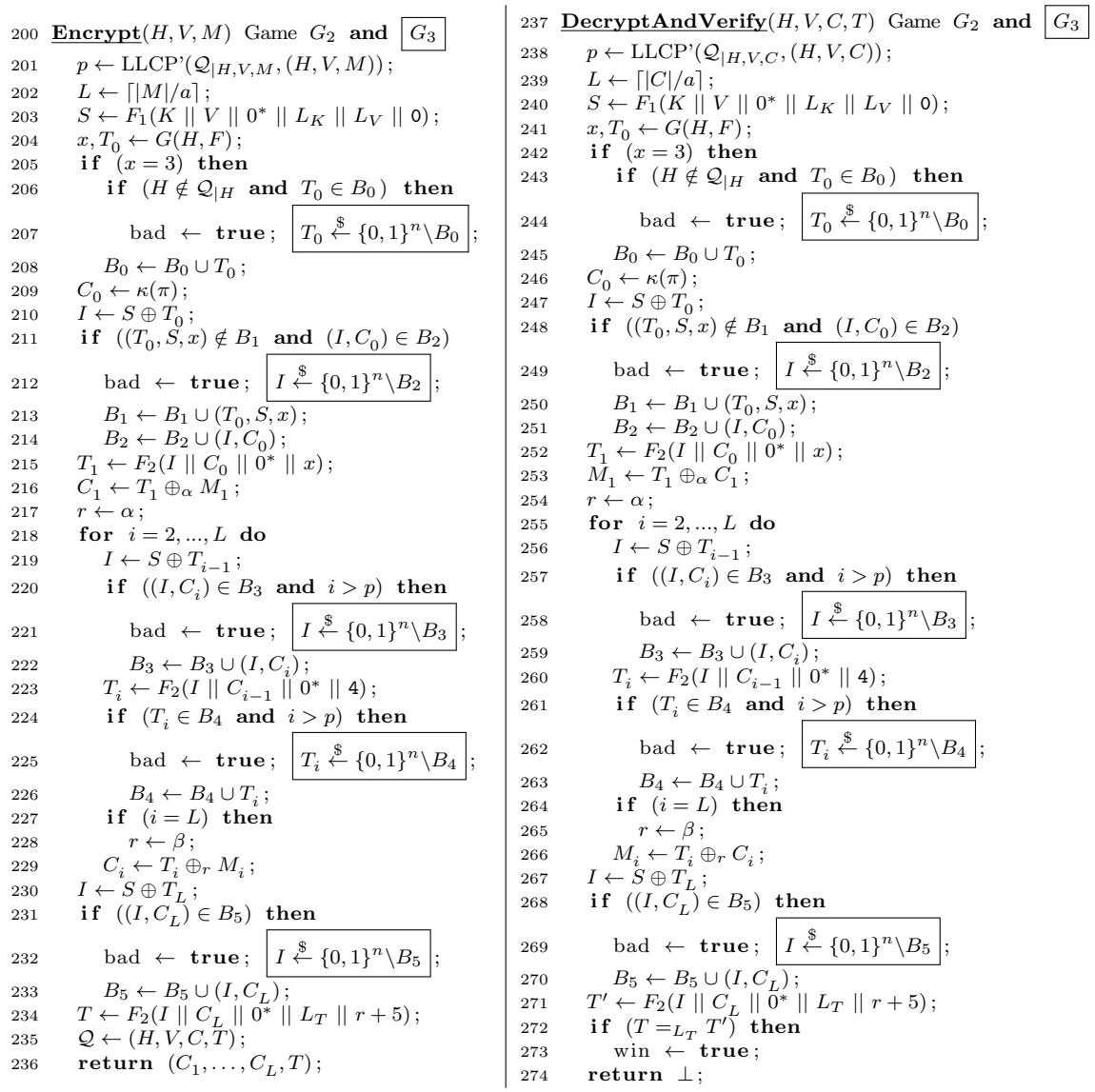

Fig. 6. Games G_2 and G_3 for the proof of Lemma 2. Game G_3 contains the code in the box while G_2 does not. The variable a denotes the block size of the message and ciphertext blocks with $a \leq n$, where n is the output size of F, F_1, and F_2, each. The function $\kappa(\pi)$ returns the first $a/4$ post decimal positions of π interpreted as a string of hex characters, and L_T denotes the bit-length of the tag.

In G_1, the encryption- and verify-placeholders are replaced by their generic **COFFE** counterparts as of Algorithm 1 and, using similar arguments as in the proof for Lemma 1, we can partition F into two independent PRF's F_1 and F_2. Thus,

$$\Pr[A^{G_0} \Rightarrow 1] \leq \Pr[A^{G_1} \Rightarrow 1] + 2 \cdot \mathbf{Adv}^{\text{PRF-XRK}}_{F_*}(q + \ell, O(t)),$$

where

$$\mathbf{Adv}^{\text{PRF-XRK}}_{F_*}(q, \ell, t) = \max\left\{\mathbf{Adv}^{\text{PRF-XRK}}_{F_2}(q + \ell, O(t)), \mathbf{Adv}^{\text{PRF}}_{F_1}(q, O(t))\right\}.$$

We now discuss the differences between G_1 and G_2. The sets $B_0, \ldots, B_5$ are initialized as empty sets (cf. Line 3 of Figure 5) and collect fresh values as follows:

- $\mathbf{B_0}$ collects all fresh values T_0, where $|H| > n$ in lines 208 and 245.
- $\mathbf{B_1}$ collects all fresh pairs (T_0, S) in lines 213 and 250.
- $\mathbf{B_2}$ collects all fresh values $I = T_0 \oplus S$ in lines 214 and 251.
- $\mathbf{B_3}$ collects all fresh pairs $(I = T_\mu \oplus S, C_\mu)$ with $1 \leq \mu \leq L - 1$, where L is the message length in blocks. This is done in lines 222 and 259.
- $\mathbf{B_4}$ collects all fresh values T_μ with $1 \leq \mu \leq L$ in lines 226 and 263.
- $\mathbf{B_5}$ collects all fresh pairs $(I = T_L \oplus S, C_L)$. This is done in lines 233 and 270.

In lines 201 and 238, the LLCP' oracle is inquired as defined in Figure 4. Finally, the variable `bad` is set to `true` if one of the if-conditions in lines 206, 211, 220, 224, 231, 243, 248, 257, 261, or 268 is `true`. *None* of these modifications affect the values returned to the adversary and therefore,

$$\Pr[A^{G_1} \Rightarrow 1] = \Pr[A^{G_2} \Rightarrow 1].$$

It follows that

$$\begin{aligned}\Pr[A^{G_2} \Rightarrow 1] &= \Pr[A^{G_3} \Rightarrow 1] + |\Pr[A^{G_2} \Rightarrow 1] - \Pr[A^{G_3} \Rightarrow 1]| \\ &\leq \Pr[A^{G_3} \Rightarrow 1] + \Pr[A^{G_3} \textit{sets } \texttt{bad}]. \end{aligned} \tag{3}$$

We now proceed to upper bound the two terms contained in (3) – in right to left order.

The success probability of Game G_3 does not differ from the success probability of Game G_2 unless one of the following cases occur, where each case causes a bad event, i.e., the variable `bad` is set to `true`. In the following, the indices i and j denote the i-th and j-th query with $1 \leq i, j \leq q$, respectively.

Case 1 (Collision – Initial Chaining Value): In lines 207 and 244 the initial chaining value T_0 is set to a new random value if the function G returns the same T_0 twice for two distinct values $H^i \neq H^j$ with $i \neq j$ and $|H^i|, |H^j| > n$, i.e., in the case when $x = 3$. The probability for such a collision can be upper bounded by

$$q^2/2^n.$$

Case 2 (Input Collision – Domain $\mathbf{1}, \ldots, \mathbf{3}$): In lines 212 and 249 the chaining value I is set to a new random value if there is a non-trivial input collision between the two input values $I^i = S^i \oplus T_0^i$ and $I^j = S^j \oplus T_0^j$ with $x^i = x^j$, so that $I^i = I^j$ with $i \neq j$. We can upper bound the success probability for this case by

$$\ell^2/2^n.$$

Case 3 (Input Collision – Domain 4): In lines 220 and 257 we test for a non-trivial input collision for the pairs $\rho_i = (S^i \oplus T^i_\mu, C^i_\mu)$ and $\rho_j = (S^j \oplus T^j_\nu, C^j_\nu)$ with $i \neq j$, $\rho_i = \rho_j$, and $1 \leq \mu, \nu \leq L - 1$. The success probability for this case can be upper bounded by

$$\ell^2/2^n.$$

Case 4 (Output Collision – Domain 4): In lines 224 and 261 we test, if the adversary has found a non-trivial collision of the form $T^i_\mu = T^j_\nu$ with $2 \leq \mu, \nu \leq L - 1$ and $(i, \mu) \neq (j, \nu)$. The success probability is then given by

$$\ell^2/2^n.$$

Case 5 (Input Collision – Domain 5): In lines 231 and 268 we test for a non-trivial input collision for the pairs $\rho_i = (S^i \oplus T^i_L, C^i_L)$ and $\rho_j = (S^j \oplus T^j_L, C^j_L)$ with $i \neq j$ and $\rho_i = \rho_j$. We can upper bound the success probability for this case by

$$q^2/2^n.$$

By adding up the individual bounds, it follows that

$$\Pr[A^{G_3} \textit{sets } \texttt{bad}] \;\leq\; \frac{3\ell^2 + 2q^2}{2^n}.$$

The adversary wins Game G_3 *iff* the variable `win` is set to `true`, i.e., the if-condition in Line 272 holds. This implies that the adversary can only win with a fresh query to the **DecryptAndVerify** oracle, which leads to $T =_{L_T} T'$, where T' is computed as shown in Line 271 and $=_{L_T}$ denotes the comparison over the L_T least significant bits. Lines 268 and 269 ensure that the input for the function F in Line 271 is always a fresh value, i.e., it was never asked before. Since F is modelled as a PRF, the probability for $T =_{L_T} T'$ can be upper bounded by

$$1/2^{L_T}.$$

As we allow the adversary to ask at most q queries, the success probability for Game G_3 can be upper bounded by

$$\Pr[A^{G_3} \Rightarrow 1] \;\leq\; q/2^{L_T}.$$

Our claim follows by adding up the individual bounds. □