The POET Family of On-Line Authenticated Encryption Schemes

Submission to the CAESAR Competition

Version 2.0
Submitted: August 29, 2015
Last edited: August 29, 2015

Farzaneh Abed	Bauhaus-Universität Weimar, `farzaneh.abed(at)uni-weimar.de`
Scott Fluhrer	Cisco Systems, `sfluhrer(at)cisco.com`
John Foley	Cisco Systems , `foleyj(at)cisco.com`
Christian Forler[1]	Huawei Technologies, `christian.forler(at)huawei.com`
Eik List	Bauhaus-Universität Weimar, `eik.list(at)uni-weimar.de`
Stefan Lucks[2]	Bauhaus-Universität Weimar, `stefan.lucks(at)uni-weimar.de`
David McGrew	Cisco Systems, `mcgrewd(at)cisco.com`
Jakob Wenzel	Bauhaus-Universität Weimar, `jakob.wenzel(at)uni-weimar.de`

The latest version of this document and all related materials can always be found on the POET homepage:
`http://www.uni-weimar.de/de/medien/professuren/mediensicherheit/research/poet/`

[1]The research leading to these results received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement no. 307952.
[2]A part of this research was done while Stefan Lucks was visiting the National Institute of Standards and Technologies (NIST).

Revision History

Changes from Version 1.3 to Version 2.0

- Added support for intermediate tags (cf. Section 5.1).
- Unified the keys of the top and bottom hash-function layers (cf. Section 5.1).
- Simplified the processing of the final block of associated data (cf. Section 5.1).
- Simplified the tag-generation step (cf. Section 5.1).
- Updated recommendations to include the parameters for intermediate tags (cf. Section 5.3).
- Revised the notions for clarity (cf. Chapter 6).
- Revised the integrity proof to the new specification and moved from the INT-CTXT to the stronger INT-RUP notion (cf. Section 7.3).
- Revised the privacy proof to the new specification (cf. Section 7.2).
- Added privacy and integrity proofs for POET with intermediate tags (cf. Sections 7.4, 7.5).
- Added performance figures for software implementation on Haswell (cf. Chapter 8).

Changes from Version 1.2 to Version 1.3

- Clarified the security goals as pointed out by Yu Sasaki.

Changes from Version 1.1 to Version 1.2

- Abandoned version of POET with Galois-Field multiplication for hashing after Abdelraheem et al. [3] pointed out the large number of weak keys.
- Acknowledged Abdelraheem et al.

Changes from Version 1.02 to Version 1.1

- Abandoned POET-m.
- Updated test vectors (cf. Appendix C).
- Added encoding conventions (cf. Section 8.1).
- Updated assumptions of Theorem 7.1 and 7.2 (cf. Section 7.1 and 7.2).
- Acknowledged Mridul Nandi for his observations on POET and POET-m.

Changes from Version 1.01 to Version 1.02

- Updated consent: Replaced “we” by “submitters” (cf. Chapter 12).
- Added a prioritized list of recommended parameter sets (cf. Section 5.3).

Executive Summary

There is a compelling need for On-Line Authenticated Encryption (OAE) schemes that are fast, secure, flexible, and robust against misuse all at the same time. This work proposes POET (***P**ipelineable **O**n-line **E**ncryption with authentication **T**ag*), a family of OAE schemes which satisfies all the mentioned properties. At its core, POET grounds on the POE family of on-line ciphers (***P**iplineable **O**n-line **E**ncryption*).

POET is fast. Its throughput is comparable to that of reference authenticated ciphers, such as OCB3 or AES-GCM, which lack the robustness provided by POET. Moreover, POET introduces a minimal overhead of only two additional block-cipher calls to generate the authentication tag. For an efficient transmission, POET adds only the tag, avoiding any overhead at the message.

POET is robust. The standard security notions for AE schemes – that POET satisfies up to the birthday bound – assumes that adversaries never repeat nonces, and do not obtain information about decrypted ciphertexts if the authentication fails. The security guarantees of almost all previous AE schemes fall apart whenever these assumptions are violated. These are a highly relevant and greatly underestimated practical issues. POET addresses them by providing security even under *both* "nonce misuse" *and* "decryption misuse".

POET is provably secure. POET bases on well-studied primitives, which simplifies the formal analysis greatly. We provide a security proof, making standard assumptions on the block cipher's security.

POET is flexible. POE and POET are ready-to-use for a variety of applications. We provide a fully generic specification to allow programmers to choose primitives that are tailored to their use case. As a recommendation, we propose the AES as block cipher, and use either four-round AES, or the full AES for universal hashing. As a desirable side effect of our recommendation, we are convinced that POET can be standardized seamlessly.

POET is efficient on a variety of platforms. POET is well-suited for low-end applications, especially when the AES is used for both encryption and universal hashing, which reduces code size and chip space. Mid-range and high-end devices can run POET efficiently thanks to pipelining. In general, software implementations benefit from the wide availability of AES native instructions on current platforms.

Contents

1. Introduction **1**

2. Features **4**

3. Security Goals **6**

4. Preliminaries **7**
- 4.1. Universal Hash Functions . . . 8
- 4.2. Block Ciphers . . . 9
- 4.3. On-Line Ciphers . . . 9
- 4.4. Authenticated Encryption Schemes . . . 10

5. Specification **14**
- 5.1. Definition of POET . . . 15
- 5.2. Instantiations for the Family of Hash Functions . . . 18
- 5.3. Recommended Parameter Sets . . . 19
- 5.4. Specification of POE . . . 19

6. Security Notions **21**
- 6.1. General Security Notions for AE Schemes . . . 21
- 6.2. Security Notions for On-Line AE Schemes . . . 23

7. Security Analysis **28**
- 7.1. OPERMCCA Security Analysis of POE . . . 28
- 7.2. OCCA Security Analysis of POET Without Intermediate Tags . . . 31
- 7.3. INT-RUP Security Analysis of POET Without Intermediate Tags . . . 33
- 7.4. OCPA-IT Security Analysis of POET With Intermediate Tags . . . 39
- 7.5. INT-CTXT-IT Security Analysis of POET with Intermediate Tags . . . 42

8. Implementation **46**
- 8.1. Encoding Conventions . . . 46
- 8.2. Software Performance of POET . . . 46

9. Design Rationale **50**

10. Acknowledgments **52**

11. Intellectual Property **53**

12. Consent **54**

Bibliography **55**

A. Lemmas of the OPERMCCA Analysis of POE **59**
A.1. Upper Bound for $\mathsf{COLL}^{\text{enc}}$ 59
A.2. Upper Bound for NOCOLLWIN 60

B. Lemmas of the OCCA Analysis of POET **62**
B.1. Upper Bound for $\mathsf{COLL}^{\text{enc}}$ 62
B.2. Upper Bound for $\mathsf{COLL}^{\text{ad}}$ 63
B.3. Upper Bound for $\mathsf{COLL}^{\text{lmb}}$ 63

C. Test Vectors for POET **67**
C.1. Four-Round AES 67
C.2. Full-Round AES 69

Chapter 1

Introduction

This document describes the on-line authenticated encryption scheme POET, and the on-line cipher POE, which serves as base for POET. Prior to our specification, we provide a brief overview.

(Nonce-Based) Authenticated Encryption. Authenticated Encryption (AE) schemes shall protect both privacy and authenticity of messages. Authenticated encryption schemes with support for Associated Data (AEAD) provide additional authentication for associated data. The standard security requirement for AE schemes is to prevent leakage of any information about secured messages except for their respective lengths. However, stateless encryption schemes would enable adversaries to detect if the same associated data and message has been encrypted before under the current key. Thus, Rogaway proposed nonce-based encryption, where the user must provide an additional nonce for every message she wants to process – a number used once (nonce).

Robustness against Nonce Misuse. In theory, the concept of nonces is simple. In practice, flawed implementations of nonces are ubiquitous [14, 30, 36, 52]. Though, apart from implementation failures, some settings render it difficult to almost impossible to prevent nonce reuse: a persistent counter, which is increased and written back every time a new nonce is needed, may be reseted by a backup, usually after some previous data loss. Similarly, the internal persistent state of an application may get duplicated when it is used as a cloned virtual machine image. Though, the majority of widely used AE schemes protect neither the confidentiality nor the integrity of messages in the case that a nonce repeats [22]. In particular, the security of all the most widely used AE schemes EAX [9], GCM [39], and OCB3 [37] falls completely apart under such circumstances. **Therefore, it appears highly desirable that a modern AE scheme can provide a second line of defense under nonce reuse.**

Robustness against Unverified Plaintext Release. The standard AE security assumes that an adversary can learn nothing about the decrypted plaintext whenever a ciphertext fails the authenticity check. This assumption is inherent in the idea of authenticated encryption and part of its strength. Though, this implies the need for buffering the entire plaintext before the authentication is verified, which is again difficult (or impossible) in some settings—may it be due to lack of memory or demanding performance requirements (e.g., high speed, low latency, and long messages).

Optical Transport Networks (OTNs) represent one example for highly demanding settings [31]. In such environments, the links between multiple network channels must be capable of transmitting, multiplexing, and switching between massive data streams in a fast and secure manner, with high throughput rates of up to 100 Gbps, allowed latencies in the magnitude of a few clock cycles, and large message frames of up to 64 kB. At that size, a mode of operation using a 128-bit block cipher would require about 4 096 block cipher invocations to complete a decryption, introducing a latency that exceeds the minimum latency goal of OTNs by far.

Theoretically, one could output the decrypted message before its authenticity was checked, which solves the latency and caching issues. Though, most AE schemes can no longer sustain neither the privacy nor the integrity of messages in the former case. As an alternative, Fouque et al. [25] proposed to mask the plaintext with an intermediate key before releasing it, and to pass the correct key to the receiver only after the message has been successfully verified. This approach solves the caching and security issues, but only shifts the burden of buffering to the next layer. Moreover, it still suffers from high latency for processing the message twice. **Therefore, a practical need for AE schemes that provide a second line of defense under decryption misuse has araised, i.e., a decent level of security, even when decryptions of non-authentic ciphertexts were compromised.**

On-Line Ciphers. (Authenticated) encryption schemes can be distinguished into such that process messages in an off-line and such that work in an on-line manner. The former require to process the plaintext at least twice to ensure that every bit of the ciphertext depends on every bit of the plaintext. On the other hand, on-line constructions split the message into blocks, such that the i-th ciphertext block depends only on the blocks $1, 2, \ldots, i$. In the accidental case when nonces are repeated, robust off-line schemes that are secure against chosen-ciphertext adversaries can provide for the potentially strongest level of security, where the adversary can detect only the repeated encryption of the same plaintext under the same key. As shown by Fleischmann et al. [22], robust on-line schemes can still ensure that the best an adversary can hope for is the detection of the longest common prefix of two messages that were encrypted under the same key and nonce. Though, on-line schemes need to process each input block only once, and therefore, allow to output ciphertexts at a significantly lower latency, which may be crucial for various applications.

Intermediate Tags. AE schemes that base on an on-line cipher that is secure against chosen-ciphertext adversaries (which we denote by OPERMCCA security, hereafter) provide an additional desirable feature: the seamless integration of intermediate authentication tags [11]. This can be achieved by adding redundancy (e.g., fixed constants or non- cryptographic checksums) to the plaintexts. For instance, the headers of IP, TCP, or UDP [42, 43, 41] packets contain a 16-bit checksum each, which is verified by the receiver and/or network routers. In OTNs, every 64-kB message frame usually consists of multiple IP packets. Due to the low-latency constraints, receiving routers are not allowed to buffer incoming messages and must forward the first packets towards their destination before the last packages have arrived – and could be checked. However, they can test the validity of the smaller, individual packets' checksums to efficiently detect forgery attempts. The definition of OPERMCCA security ensures that the first TCP/IP packet with an invalid CRC-16 checksum only passes with a probability of at most 2^{-16}. Even if this packet passes, the next packet would again only pass with the same probability and so on and so forth.

POET in a Nutshell. This work introduces the first non-sequential robust[1] on-line AE scheme, called *Pipelineable On-line Encryption with authentication Tag* (or POET hereafter), which is based on an OPERMCCA-secure family of on-line ciphers, called *Pipelineable On-line Encryption* (POE). POE and POET consist of an ECB layer that is protected by two chaining layers with an ϵ-AXU family of hash functions. The property of pipelineability distinguishes POET from previous CCA-secure on-line ciphers (e.g., TC3 [50]), which are inherently sequential. Thus, POET provides a significantly higher throughput on multi-core architectures with integrated AES native instructions, and also allows to utilize single-core processors more efficiently.

[1]By robust, we mean to preserve integrity and on-line privacy against both nonce misuse and decryption misuse.

We define POE and POET in a generic way, allowing the user to choose well-suitable instances for the cipher and the hash function. We propose two instances which all using the AES-128 as block cipher, and using either four-round AES-128, or the full AES-128 for universal hashing.

Outline. The remainder of this work is structured as follows. In Chapter 2, we give a brief overview over our design goals. Security goals are discussed in Chapter 3. Next, Chapter 4 recalls the necessary preliminaries about universal hash functions, on-line ciphers, and AE schemes that are used in the subsequent parts of this work. Chapter 5 contains the specification of POE and POET. In Chapter 6, we define the relevant security notions used in our work. Chapter 7 is devoted to the security analysis. Next, Chapter 8 provides details on implementational aspects -and performance evaluation of POET. Chapter 9 contains our design rationale. Finally, Chapters 10, 11, and 12 contain acknowledgments and the obligational statements regarding intellectual property and consent.

Chapter 2

Features

Length-Preserving Encryption. POET processes messages of arbitrary lengths in a length-preserving manner, i.e., it encrypts m-bit plaintexts to m-bit ciphertexts, without appending any padding, which is particularly useful for (battery-powered) resource-constrained devices, where the transmission of additional bits is costly.

On-Line. POET provides on-line encryption and decryption, i.e., it can process the i-th input block before the $(i+1)$-th block has been read.

Authentication of Associated Data of Arbitrary Lengths. POET allows to authenticate associated data (or header, hereafter) of arbitrary lengths, including the empty string. Since the result of the header-processing step is required as an input parameter for the tag-generation process, POET appends the public message number and pads the given header with a standard 10* padding. Thus, the POET approach renders the entire header into a nonce.

In theory, POET could employ any secure MAC to process the header. Among the variety of existing constructions, we *borrow* the provably secure PMAC1 design which allows to process the header blocks in arbitrary and parallelizable order to reduce the latency on multi-core CPUs.

Support For Intermediate Tags. POET offers built-in support for intermediate tags when messages already contain some well-formed redundancy, e.g., fixed constants or non-cryptographic checksums. Therefore, POET is well-suited for low-latency environments, such as OTNs, where messages usually consist of multiple TCP/IP packages with integrated (although small) checksums. Note that non-cryptographic intermediate tags lack the level of security of cryptographic authentication tags.

Variable Tag Lengths. While we recommend tags of the block cipher's state size n, POET also provides limited support for truncated tags. Network protocols – such as TLS 1.x [18, 19, 20] or IPSec [1, 35] – usually employ authentication tags of 96 bits. For messages whose length is a multiple of n, POET provides full flexibility to choose tag sizes. In the other cases, tags can still be truncated but only with the requirement that tag and final message block should sum up to at least n bits. Note that this complies with the TLS and IPSec protocol suites.

Support of Static Associated Data. POET allows the result of the header-processing step to be cached and reused for subsequent messages.

Performance. Our recommended instance of POET uses the full AES-128 as cipher and four-round AES as universal hash function. Therefore, POET can benefit greatly from the available AES native instruction sets of current processors. PMAC1 provides POET with a maximum of parallelism when the header is processed. For the message encryption and decryption, POET requires only a single block-cipher and two hash-function calls per message block. The non-sequential design of POET allows to efficiently process subsequent message blocks exploiting the CPU pipeline and multi-threading techniques.

Comparison with AES-GCM.

POET has the following advantages over AES-GCM:

- **Nonce-misuse resistance.** AES-GCM is completely insecure whenever a nonce repeats. POET provides on-line privacy and full integrity in this case.
- **Decryption-misuse resistance.** AES-GCM is completely insecure against decryption misuse. Interfaces that decrypt and output block by block before checking the final tag leak information about the plaintexts. POET ensures that the best that an adversary sees are longest common prefixes to other messages.
- **Support for intermediate tags.** For resource-constrained environments, POET supports intermediate tags that can be checked before outputting the plaintext.
- **Arbitrary tag lengths.** AES-GCM loses security when tags are truncated to other lengths than 128 bits. POET allows tags of arbitrary lengths.
- **No weak keys.** As highlighted by Saarinen [51], Procter and Cid [44], Zhu, Tan, and Gong [57], and Abdelraheem et al. [2], almost all subsets of keys in AES-GCM are weak. The recommended instances of POET use only the AES which is unlikely to have weak keys.
- **Single primitive.** POET uses only the AES as a primitive. There is no need for an implementation of Galois-Field multiplications, which is beneficial especially for saving chip area. The doubling operations in the header-processing step of POET can be implemented with simple XOR and shift operations.
- **No restrictions from patents on AES-GCM.**

On the other hand, AES-GCM possesses the following advantages over POET:

- **Full parallelizability.** The hash-function layer of POET is partially sequential.
- **Integrity security up to $O(2^n)$ queries.** Due to its structure, POET can provide integrity only up to the birthday-bound number of queries.
- **Inverse-freeness.** Due to the use of counter mode, AES-GCM does not require the inverse operation of the block cipher. The decryption operation of POET requires also the decryption operation of the cipher to be implemented. Though, devices that only need to encrypt require only the forward operation.
- **Asynchronous processing of associated data and message.** AES-GCM allows to encrypt the message before or parallel to processing the associated data. POET uses the result of the header-processing step as input to its encryption (or decryption) operation.

Chapter 3

Security Goals

For all our recommended instantiations of POET, we claim a security level close to 128 bits for all attacks if the data complexity is $\ll \epsilon^{-\frac{1}{2}}$ blocks, i.e., $\ll 2^{56}$ blocks for POET-AES10-AES4, and $\ll 2^{64}$ blocks for POET-AES10-AES10. For details see Table 3.1.

	Bits of Security
Confidentiality for the plaintext	$\log_2(2^{128} - c \cdot \epsilon \cdot \ell^2)$
Integrity for the plaintext	$\log_2(2^{128} - c \cdot \epsilon \cdot \ell^2)$
Integrity for the associated data	$\log_2(2^{128} - c \cdot \epsilon \cdot \ell^2)$
Integrity for the public message number	$\log_2(2^{128} - c \cdot \epsilon \cdot \ell^2)$
Security against key recovery	128
Security against tag guessing	128

Table 3.1.: Claimed bits of security for all our recommended instantiations of POET; ℓ denotes the data complexity and c a constant that can be deferred from our security results in Chapters 7; $\epsilon \approx 2^{-113}$ for POET with four-round AES; $\epsilon \approx 2^{-128}$ for POET with full AES.

POET does not intent to support secret message numbers, i.e., the length of secret message numbers is 0 bits. Our recommended instantiations of POET all use the AES-128 as block cipher. Hence, the block size for message blocks, header blocks, and nonce is 128 bits. The recommended tag size for all recommended instantiations is 128 bits.

POET is designed to provide robustness against nonce misuse, i.e., POET maintains full integrity and confidentiality, except for leaking collisions of the longest common prefix of messages. Furthermore, it provides robustness against decryption misuse, where it maintains confidentiality up to detection of the longest common prefix of messages that were previously encrypted under the same key and with the same header and nonce.

Note that the CAESAR call for submissions specifies "bits of security" as "the logarithm base 2 of the attack cost". We understand this as the cost for the attacker winning the attack game with significant probability, typically 50 %. We do not consider attacks with insignificant probability p even if $p >$ attack cost$/2^{128}$.

Chapter 4

Preliminaries

This document describes the on-line authenticated encryption scheme POET, and the on-line cipher POE, which serves as base for POET. This section introduces the general notions that are used throughout this work. Table 4.1 summarizes the most frequently used identifiers.

Identifier	**Description**
C	Ciphertext
E/E^{-1}	Cipher (encryption function)/Inverse cipher
F	Function, mostly universal hash function
H	Header (= associated data)
K	Cipher key
L	Key for header processing
K_F	Key for the ϵ-AXU family of hash functions F
M	Plaintext message
N	Public message number (= initial value/nonce)
SK	User-provided secret key
T	Authentication tag
τ	Result of the header-processing procedure
n	Block length in bits
k	Key length in bits
$\|X\|$	Length of X in bits
$\langle X \rangle_x$	Encoding of an integer X as x-bit little-endian string
X_i	i-th block of a value X
X^i	The i-th part of a partitioned string
$X \,\|\|\, Y$	Concatenation of two values X and Y
$X \,\|\|\, 10^*$	Value X with a single '1'-bit appended, and then padded with zeros until its length is a multiple of n
$\mathcal{X}$	Set or family $\mathcal{X}$
$X \twoheadleftarrow \mathcal{X}$	X is a uniformly at random chosen sample from $\mathcal{X}$.
$\mathcal{A}^{\mathcal{O}}$	An adversary that interacts with an oracle $\mathcal{O}$.
$\mathcal{O}_1 \hookrightarrow \mathcal{O}_2$	The output of oracle $\mathcal{O}_1$ is given as input to oracle $\mathcal{O}_2$.
ε	The empty string.

Table 4.1.: Notions used throughout this paper.

In general, we write uppercase letters (X, Y) to denote functions, parameters, or values; lowercase letters (x, y) to denote lengths; and calligraphic uppercase letters ($\mathcal{X}$, $\mathcal{Y}$) to represent sets or families of functions (e.g., $\mathcal{F}$). For convenience, we introduce a notation for a *restriction of a set.* Let $\mathcal{Q} = \{0,1\}^a \times \{0,1\}^b \times \{0,1\}^c$, then we denote by $\mathcal{Q}_{|b,c} = \{(B,C) \mid \exists A : (A,B,C) \in \mathcal{Q}\}$ as a restriction of $\mathcal{Q}$ to B and C. This generalizes in the obvious way.

An adversary $\mathcal{A}$ is an efficient Turing machine that interacts with a given set of oracles, which appear as black boxes to $\mathcal{A}$. Wlog., we always assume a deterministic adversary. We use the notation $\mathcal{A}^{\mathcal{O}}$ for the output of $\mathcal{A}$ after interacting with some oracle $\mathcal{O}$. We will provide pseudo-code descriptions of the oracles, which will be referred to as games, according to the game-playing framework by Bellare and Rogaway [8]. Each game consists of a set of procedures and an adversary that interacts with the oracles by calling these procedures.

4.1. Universal Hash Functions

We use well-studied properties of universal hash-function families. This section recalls the relevant standard definitions from the literature by Carter and Wegman [15, 56], the theorem related to these functions by Boesgaard et al. [13], and their composition by Stinson [53, 54].

Definition 4.1 (ϵ-Almost-(XOR-)Universal Hash Functions).
Let $\mathcal{F} = \{F \,|\, F : \{0,1\}^m \to \{0,1\}^n\}$ denote a family of hash functions. $\mathcal{F}$ is called ϵ-almost-universal *(ϵ-AU) iff for all $X, X' \in \{0,1\}^m$, $X \neq X'$:*

$$\Pr[F \twoheadleftarrow \mathcal{F} :\ F(X) = F(X')] \leq \epsilon.$$

$\mathcal{F}$ is called ϵ-almost-XOR-universal *(ϵ-AXU) iff for all $X, X' \in \{0,1\}^m$, $Y \in \{0,1\}^n$, $X \neq X'$:*

$$\Pr[F \twoheadleftarrow \mathcal{F} :\ F(X) \oplus F(X') = Y] \leq \epsilon.$$

The definition of strong universal hash functions is similar.

Definition 4.2 (Strong Universal Hash Functions).
Let $\mathcal{F} = \{F \,|\, F : \{0,1\}^m \to \{0,1\}^n\}$ denote a family of hash functions. $\mathcal{F}$ is called strongly universal *(SU) iff for all $X \in \{0,1\}^m, Y \in \{0,1\}^n$:*

$$\Pr[F \twoheadleftarrow \mathcal{F} :\ F(X) = Y] \leq 1/2^n,$$

and for all $X, X' \in \{0,1\}^m$ with $X \neq X'$ and $Y, Y' \in \{0,1\}^n$:

$$\Pr[F \twoheadleftarrow \mathcal{F} :\ F(X) = Y, F(X') = Y'] \leq 1/2^{2n}.$$

Boesgaard et al. showed in [13] that an ϵ-AXU family of hash functions can be reduced to a family of ϵ-AU hash functions by XORing an arbitrary value to the output:

Theorem 4.3 (Theorem 3 from [13]). *Let $\mathcal{F} = \{F : \{0,1\}^m \to \{0,1\}^n\}$ be a family of ϵ-AXU hash functions. Then, the family $\mathcal{F}' = \{F' : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^n\}$ with $F'(X,Y) = F(X) \oplus Y$ is ϵ-AU.*

The effects of composing two universal hash function instances were studied by Stinson in [53, 54].

Theorem 4.4 (Theorem 5.4 from [54]). *Let $\mathcal{F} = \{F \,|\, F : \{0,1\}^m \to \{0,1\}^n\}$ be an ϵ_1-AU family of hash functions and $\mathcal{G} = \left\{G \,|\, G : \{0,1\}^n \to \{0,1\}^\ell\right\}$ an ϵ_2-AU family hash functions. Then, there exists an ϵ-AU family of hash functions $\mathcal{H}$ with $\epsilon \leq \epsilon_1 + \epsilon_2$ and $|H| = |\mathcal{F}| \times |\mathcal{G}|$.*

4.2. Block Ciphers

A block cipher is a keyed family of n-bit permutations $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$, which takes a k-bit key K and an n-bit message M, and outputs an n-bit ciphertext C. We denote $\texttt{Block}(k, n)$ as the set of all (k, n)-bit block ciphers for $n > 0$. For any $E \in \texttt{Block}(k, n)$ and a fixed key $K \in \{0,1\}^k$, the encryption of a message M is given by $E_K(M)$, and the decryption is defined as the inverse function, i.e., $E_K^{-1}(M)$. For any key $K \in \{0,1\}^k$, it applies that $E_K^{-1}(E_K(M)) = M$.
We define the IND-SPRP-security of a block cipher E by the success probability of an adversary trying to differentiate between the block cipher and an n-bit random permutation $\pi(\cdot)$. We denote by Perm_n the set of all n-bit permutations.

Definition 4.5 (IND-SPRP-Security). *Let $E \in \texttt{Block}(k, n)$ denote a block cipher and E^{-1} its inverse. Let Perm_n be the set of all n-bit permutations. The IND-SPRP advantage of $\mathcal{A}$ against E is then defined by*

$$\mathbf{Adv}_{E,E^{-1}}^{\textsf{IND-SPRP}}(\mathcal{A}) \leq \left|\Pr\left[\mathcal{A}^{E(\cdot),E^{-1}(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\pi(\cdot),\pi^{-1}(\cdot)} \Rightarrow 1\right]\right|,$$

where the probabilities are taken over $K \twoheadleftarrow \{0,1\}^k$ and $\pi \twoheadleftarrow \mathsf{Perm}_n$. We define $\mathbf{Adv}_{E,E^{-1}}^{\textsf{IND-SPRP}}(q, t)$ as the maximum advantage over all IND-SPRP-adversaries $\mathcal{A}$ on E that run in time at most t and make at most q queries to the available oracles.

4.3. On-Line Ciphers

Definition 4.6 (On-Line Cipher). *Let $\Gamma : \{0,1\}^k \times (\{0,1\}^n)^* \to (\{0,1\}^n)^*$ denote a keyed family of n-bit permutations, which takes a k-bit key K and a message M of an arbitrary number of n-bit blocks, and outputs a ciphertext C consisting of the same number of n-bit blocks as M. We call Γ an* on-line cipher *iff the encryption of message block M_i, for all $i \in [1, |M|/n]$, depends only on the blocks $M_1, \ldots, M_i$.*

Usually, a secure cipher that processes messages of arbitrary lengths should behave like a random permutation. It is easy to see that on-line ciphers are in conflict with this security property since the encryption of message block M_i does not depend on M_{i+1}. The on-line behavior implies that two messages M and M' with an m-block common prefix will always be encrypted to two ciphertexts C and C', which also share an m-block common prefix. Hence, we define an on-line cipher Γ to be secure if and only if no ciphertext reveals any further information about a plaintext than its length and the *longest common prefix* with previous messages. We recall the formal definition of the length of the longest common prefix of a message from [22].

Definition 4.7 (Length of Longest Common Prefix). *For integers $n, \ell, d \geq 1$, let $\mathcal{D}_n^d = (\{0,1\}^n)^d$ denote the set of all strings that consist of exactly d blocks of n bits each. Further, let $\mathcal{D}_n^* = \bigcup_{d \geq 0} \mathcal{D}_n^d$ denote the set which consists of all possible n-bit strings and $\mathcal{D}_{\ell,n} = \bigcup_{0 \leq d \leq \ell} \mathcal{D}_n^d$ the set of all possible strings which consist of 0 to ℓ n-bit blocks. For arbitrary $P \in \mathcal{D}_n^d$, let P_i denote the i-th block for all $i \in 1, \ldots, d$. For $P, R \in \mathcal{D}_n^*$, we define the* length of the longest common prefix of n-bit blocks *of P and R by*

$$\mathsf{LLCP}_n(P, R) = \max_i \{\forall j \in 1, \ldots, i : \ P_j = R_j\}.$$

For a non-empty set $\mathcal{Q}$ of strings in D_n^*, we define

$$\mathsf{LLCP}_n(\mathcal{Q}, P) = \max_{q \in \mathcal{Q}} \{\mathsf{LLCP}_n(q, P)\}.$$

For any two distinct ℓ-block inputs M and M' that share an exactly m-block common prefix $M_1 \parallel \ldots \parallel M_m$, the corresponding outputs $C = P(M)$ and $C' = P(M')$ satisfy $C_i = C'_i$ for all $i \in [1, m]$ and $m \leq \ell$, where P denotes an on-line permutation. However, it applies that $C_{m+1} \neq C'_{m+1}$ and all further blocks C_i and C'_i, with $i \in [m+2, \ell]$, are *independent*. This behavior is defined by on-line permutations. We recall their definition in the following.

Definition 4.8 (On-Line Permutation). *Let $F_i : (\{0,1\}^n)^i \to \{0,1\}^n$ be a family of indexed n-bit permutations, i.e., for a fixed index $j \in (\{0,1\}^n)^{i-1}$ it applies that $F_i(j, \cdot)$ is a permutation. We define an n-bit on-line permutation $P : (\{0,1\}^n)^\ell \to (\{0,1\}^n)^\ell$ as a composition of ℓ permutations $F_1 \cup F_2 \cup \cdots \cup F_\ell$. An ℓ-block message $M = (M_1, \ldots, M_\ell)$ is mapped to an ℓ-block output $C = (C_1, \ldots, C_\ell)$ by*

$$C_i = F_i(M_1 \parallel \ldots \parallel M_{i-1}, M_i), \quad \forall i \in [1, \ell].$$

We denote by OPerm_n the set of all n-bit on-line permutations. Note that a random on-line permutation can be implemented efficiently by lazy sampling.

4.4. Authenticated Encryption Schemes

An authenticated encryption scheme with associated data (AEAD scheme) provides encryption for a message M and authentication for M and associated data H. We concern string-based associated data that we also denote as a header. Our definition implies that the header H contains a nonce at its end. An adversary that never repeats a nonce over all its encryption queries is called *nonce-respecting*, and *nonce-ignoring* otherwise.

Definition 4.9 (AEAD Scheme). *An authenticated encryption scheme (with associated data) is a tuple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with an encryption algorithm $\mathcal{E}_K(H, M)$ and a decryption algorithm $\mathcal{D}_K(H, C, T)$. $K \in \mathcal{K}$ denotes the key, $H \in \mathcal{H}$ the associated data (or header), $M \in \mathcal{M}$ the message, $T \in \mathcal{T}$ the authentication tag, and $C \in \mathcal{C}$ the ciphertext, where $\mathcal{K} \subseteq \{0,1\}^k$, $\mathcal{H}, \mathcal{M}, \mathcal{C} \subseteq \{0,1\}^*$, and $\mathcal{T} \subseteq \{0,1\}^t$ denote the key, header, message, ciphertext, and tag space, respectively, with $k, t > 1$. We write*

$$\mathcal{E} : \mathcal{K} \times \mathcal{H} \times \mathcal{M} \to \mathcal{C} \times \mathcal{T},$$
$$\mathcal{D} : \mathcal{K} \times \mathcal{H} \times \mathcal{C} \times \mathcal{T} \to \mathcal{M} \cup \{\perp\},$$

to state that $\mathcal{E}$ always outputs a ciphertext C and the authentication tag T for the tuple $(H,\ M)$ under a key K, and $\mathcal{D}$ outputs the decryption of $(H,\ C)$ iff the given tag is valid or $\perp$ otherwise. The correctness condition applies that $\mathcal{D}_K(\mathcal{E}_K(H, M)) = M$ to hold for each triple (K, H, M). We call an AE scheme $\Pi = (\mathcal{E}, \mathcal{D})$ an on-line authenticated encryption (OAE) scheme if and only if $\mathcal{E}$ encrypts plaintexts in an on-line manner.

Authenticated Encryption under Unverified Plaintext Release. To analyze the security under release of unverified plaintexts, Andreeva et al. [4] proposed the notion of separated AEAD schemes.

Definition 4.10 (Separated AEAD Scheme [4]). *A separated authenticated encryption scheme (with associated data) is a tuple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{V})$ with an encryption algorithm $\mathcal{E}_K(H, M)$, a decryption algorithm $\mathcal{D}_K(H, C, T)$, and a verification algorithm $\mathcal{V}_K(H, C, T)$. $K \in \mathcal{K}$ denotes the key, $H \in \mathcal{H}$ the associated data (or header), $M \in \mathcal{M}$ the message, $T \in \mathcal{T}$ the authentication tag, and $C \in \mathcal{C}$ the ciphertext, where $\mathcal{K} \subseteq \{0,1\}^k$, $\mathcal{H}, \mathcal{M}, \mathcal{C} \subseteq \{0,1\}^*$, and $\mathcal{T} \subseteq \{0,1\}^t$ denote the key, header, message, ciphertext, and tag space, respectively, with $k, t > 1$. We write*

$$\begin{aligned}
\mathcal{E} &: \mathcal{K} \times \mathcal{H} \times \mathcal{M} \to \mathcal{C} \times \mathcal{T}, \\
\mathcal{D} &: \mathcal{K} \times \mathcal{H} \times \mathcal{C} \times \mathcal{T} \to \mathcal{M}, \\
\mathcal{V} &: \mathcal{K} \times \mathcal{H} \times \mathcal{C} \times \mathcal{T} \to \{\textbf{\textit{true}}, \perp\}.
\end{aligned}$$

*to state that $\mathcal{E}$ always outputs a ciphertext C and the authentication tag T for the tuple $(H,\ M)$ under a key K, $\mathcal{D}_K(H, C, T)$ always returns some message $M \in \mathcal{M}$, and $\mathcal{V}_K(H, C, T)$ returns **true** if (H, C, T) is valid and $\perp$ otherwise.*

On-Line Authenticated Encryption with Intermediate Tags. Intermediate tags are an effective means to allow an on-line AE scheme to output authenticated parts of the decrypted plaintext before the entire ciphertext has been processed.

Our approach to augment a given on-line AE scheme with support for intermediate tags is to wrap it by an encoding layer: Before encrypting a given header-message tuple, the AE scheme splits the message into parts, adds redundancy to each part, and encrypts them sequentially. The final ciphertext part is augmented with the usual authentication tag. During decryption, the expected redundancy can then be verified so that the i-th part can be released already after i parts have been processed. So, valid parts can be released and invalid ciphertexts can be rejected potentially much earlier than for conventional AE schemes, which may reduce latency significantly. Moreover, the implementation and security of the scheme with intermediate tags can rely on the existing components and the security of the underlying on-line AE scheme.

We borrow (in slightly modified form) the definitions τ-expanding functions from Hoang et al. [29]. We define a partitioned string $M \in (\{0,1\}^x)^*$ as a vector $M = (M^1, M^2, \ldots)$ of strings $M^i \in \{0,1\}^x$ for some fixed $x \geq 1$. We call each of its components a part. We further define a τ-expanding function $F : \{0,1\}^* \to \{0,1\}^*$ as an injective function, satisfying for all inputs $M \in \{0,1\}^*$ that $|F(M)| = |M| + \tau$.

Compared to conventional AE schemes, an on-line AE scheme with intermediate tags takes two additional integer parameters $\ell_s, \ell_t \geq 1$, where ℓ_s defines the number of subsequent message blocks that define a part, and ℓ_t denotes the expansion in bits of each part after encryption. Moreover, for a given n, an on-line AE scheme with intermediate tags fixes a family of ℓ_t-expanding functions with signature $\textbf{EncodePart}_{\ell_s,\ell_t} : \{0,1\}^{\ell_s \cdot n} \to \{0,1\}^{\ell_s \cdot n + \ell_t}$, such that each element of the family

$\textbf{EncodePart}_{\ell_s,\ell_t}$ encodes ℓ_t bits of redundancy into a given message part. $\textbf{EncodePart}_{\ell_s,\ell_t}$ defines implicitly a corresponding function $\textbf{DecodePart}_{\ell_s,\ell_t} : \{0,1\}^{\ell_s \cdot n + \ell_t} \to \{0,1\}^{\ell_s \cdot n} \cup \{\bot\}$ as

$$\textbf{DecodePart}_{\ell_s,\ell_t}(Y) = \begin{cases} X & \text{if } \exists X \in \{0,1\}^{\ell_s \cdot n} \text{ such that } \textbf{EncodePart}_{\ell_s,\ell_t}(X) = Y, \\ \bot & \text{otherwise.} \end{cases}$$

Since each function $\textbf{EncodePart}_{\ell_s,\ell_t}$ is injective, the same follows for $\textbf{DecodePart}_{\ell_s,\ell_t}$, for all $\ell_s, \ell_t \geq 0$. This means, every input Y to **DecodePart** is mapped to at most one output X. The injectivity also implies correctness, i.e., for all $X \in \{0,1\}^{\ell_s \cdot n}$ holds that

$$\textbf{DecodePart}_{\ell_s,\ell_t}(\textbf{EncodePart}_{\ell_s,\ell_t}(X)) = X.$$

We define the transformation of a given separated AEAD scheme $\Pi = (\mathcal{E}, \mathcal{D}, \mathcal{V})$ to a on-line AE scheme with intermediate tags $\widetilde{\Pi} = (\widetilde{\mathcal{E}}, \widetilde{\mathcal{D}})$ below.

Definition 4.11 (On-Line AEAD Scheme with Intermediate Tags). *Let $n, k, t \geq 1$ be fixed. Let $\textbf{\textit{EncodePart}}_{\ell_s,\ell_t} : \{0,1\}^{\ell_s \cdot n} \to \{0,1\}^{\ell_s \cdot n + \ell_t}$ be a family of ℓ_t-expanding functions for $\ell_s, \ell_t \geq 0$. Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{V})$ be a given separated on-line AE scheme (with associated data). Then, the on-line AEAD scheme with intermediate tags $\widetilde{\Pi} = (\mathcal{K}, \widetilde{\mathcal{E}}, \widetilde{\mathcal{D}})$ is a tuple of deterministic algorithms with signatures:*

$$\widetilde{\mathcal{E}} : \mathbb{N} \times \mathbb{N} \times \mathcal{K} \times \mathcal{H} \times \mathcal{M} \to \mathcal{C} \times \mathcal{T}$$
$$\widetilde{\mathcal{D}} : \mathbb{N} \times \mathbb{N} \times \mathcal{K} \times \mathcal{H} \times \mathcal{C} \times \mathcal{T} \to \mathcal{M},$$

which are defined in Algorithm 1. $\mathcal{K} \subseteq \{0,1\}^k$, $\mathcal{H}$, $\mathcal{M}$, $\mathcal{C}$, and $\mathcal{T} \subseteq \{0,1\}^t$ denote the key, header, state, message, ciphertext, and tag space, respectively.

Algorithm 1 The operations $\widetilde{\mathcal{E}}$ and $\widetilde{\mathcal{D}}$ to transform a given separated AEAD scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{V})$ to an on-line AEAD scheme with intermediate tags $\widetilde{\Pi} = (\mathcal{K}, \widetilde{\mathcal{E}}, \widetilde{\mathcal{D}})$. $\ell_s, \ell_t, x \geq 1$ are fixed integers. The definition of the family of functions $\textbf{EncodePart}_{\ell_s,\ell_t}$ is specific to the scheme; that of the family of functions $\textbf{DecodePart}_{\ell_s,\ell_t}$ is given above.

```
$\widetilde{\mathcal{E}}_K(\ell_s, \ell_t, H, M)$
101: $\widetilde{H} \leftarrow (\langle \ell_s \rangle_x \,||\, \langle \ell_t \rangle_x \,||\, H)$
102: $(\widetilde{M}^1, \ldots, \widetilde{M}^\mu) \leftarrow \textbf{Encode}_{\ell_s,\ell_t}(M)$
103: $\widetilde{M} \leftarrow (\widetilde{M}^1 \,||\, \ldots \,||\, \widetilde{M}^\mu)$
104: $(C, T) \leftarrow \mathcal{E}_K(\widetilde{H}, \widetilde{M})$
105: return $(C, T)$

$\textbf{Encode}_{\ell_s,\ell_t}(M)$
201: $\mu \leftarrow \lceil |M| / (\ell_s \cdot n) \rceil$
202: $(M^1, \ldots, M^\mu) \leftarrow \textbf{SplitInto}_{\ell_s \cdot n}(M)$
203: for $i \leftarrow 1, \ldots, \mu - 1$ do
204:     $\widetilde{M}^i \leftarrow \textbf{EncodePart}_{\ell_s,\ell_t}(M^i)$
205: $\widetilde{M}^\mu \leftarrow M^\mu$
206: return $(\widetilde{M}^1, \ldots, \widetilde{M}^\mu)$

$\widetilde{\mathcal{D}}_K(\ell_s, \ell_t, H, C, T)$
301: $\widetilde{H} \leftarrow (\langle \ell_s \rangle_x \,||\, \langle \ell_t \rangle_x \,||\, H)$
302: $\widetilde{M} \leftarrow \mathcal{D}_K(\widetilde{H}, C, T)$
303: $(M^1, \ldots, M^j) \leftarrow \textbf{Decode}_{\ell_s,\ell_t}(\widetilde{M})$
304: if $j = \mu$ and $\mathcal{V}_K(\widetilde{H}, C, T) = \bot$ then
305:     return $(M^1 \,||\, \ldots \,||\, M^{j-1})$
306: return $(M^1 \,||\, \ldots \,||\, M^j)$

$\textbf{Decode}_{\ell_s,\ell_t}(\widetilde{M})$
401: $\mu \leftarrow \lceil |\widetilde{M}| / (\ell_s \cdot n + \ell_t) \rceil$
402: $(\widetilde{M}^1, \ldots, \widetilde{M}^\mu) \leftarrow \textbf{SplitInto}_{\ell_s \cdot n + \ell_t}(M)$
403: for $i \leftarrow 1, \ldots, \mu - 1$ do
404:     $M^i \leftarrow \textbf{DecodePart}_{\ell_s,\ell_t}(\widetilde{M}^i)$
405:     if $M^i = \bot$ then
406:         return $(M^1, \ldots, M^{i-1})$
407: return $(M^1, \ldots, M^\mu)$
```

For any fixed key $K \in \mathcal{K}$, header $H \in \mathcal{H}$, parameters $\ell_s, \ell_t \geq 0$, and messages $M \in \mathcal{M}$, it holds that $\widetilde{\mathcal{D}}_K(H, \widetilde{\mathcal{E}}_K(H, M)) = M$. For domain separation, we define that the parameters ℓ_s and ℓ_t are encoded as x-bit strings, for a scheme-specific predefined $x \geq 1$, into the header. We denote this encoding to x-bit strings by $\langle \ell_s \rangle_x$ and $\langle \ell_t \rangle_x$. The function **SplitInto** : $\mathbb{N} \times \{0,1\}^* \to \{0,1\}^{**}$ splits a given string $X \in \{0,1\}^*$ to a partitioned string $(X^1, \ldots, X^{\mu-1}, X^{\mu})$, such that the length of each part except the last is $|X^1| = \ldots = |X^{\mu-1}| = x$ bits, the length of the last part $|X^{\mu}| \leq x$, and it holds that $X = X^1 \| \ldots \| X^{\mu}$.

Note that at least three strongly related approaches exist in literature. Datta and Nandi [17] proposed an approach for ELmD, which exploits the fact that ELmD transforms the i-th internal state value differently for computing the next output block and for updating the internal state. It may be worth investigating how to generalize their approach to non-EME designs.

Hoang et al. [28, 29] describe a very general approach to on-line AE which is strongly related to intermediate tags. Their strategy splits the message into variable-length parts and encrypts each part with a robust offline authenticated encryption scheme in order to reduce latency at decryption. The authors devise the OAE2 notion for their approach, renaming conventional on-line authenticated encryption as OAE1, and propose fine-grained notions OAE2a, OAE2b, OAE2c, nOAE, and dOAE. Moreover, Bertoni et al. [11] proposed the duplexing method for sponge-based AE schemes, which is similar to OAE2.

Chapter 5

Specification

This chapter defines the POET family of on-line AE schemes. From a top-level point of view, POET consists of three layers:

1. The top-row layer applies an ϵ-AXU function F_{K_F} to the previous top-row chaining value X_{i-1} and computes the XOR of the output of F_{K_F} and the message block M_i: $X_i = M_i \oplus F_{K_F}(X_{i-1})$.
2. The middle layer encrypts the current top-row chaining value: $Y_i = E_K(X_i)$.
3. The bottom-row layer applies the ϵ-AXU function F_{K_F} to the previous bottom-row chaining value Y_{i-1} and computes the XOR of the output of F_{K_F} and the encrypted block. The result denotes the ciphertext block: $C_i = Y_i \oplus F_{K_F}(Y_{i-1})$.

A schematic illustration of the encryption process of POET is given in Figure 5.1.

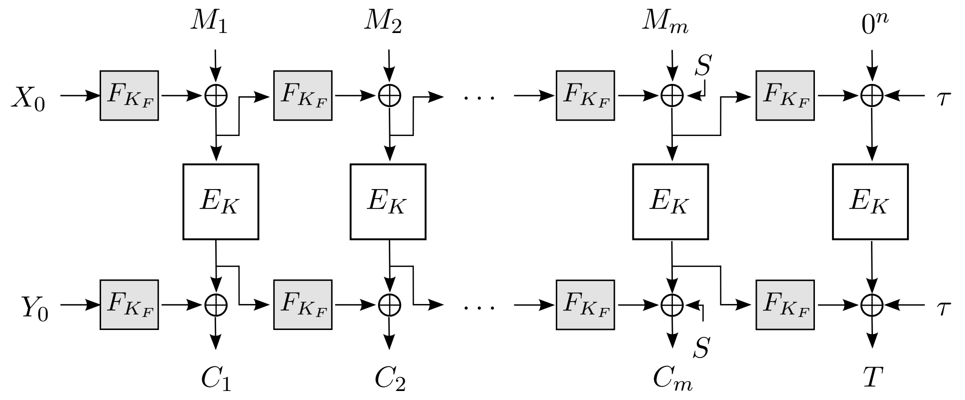


Figure 5.1.: Schematic illustration of the encryption process of POET for an m-block message $M = (M_1, \ldots, M_m)$. F_{K_F} denotes a family of ϵ-AXU hash functions and E a block cipher; $S = E_K(|M|)$ denotes the encrypted message length, τ the result of the header-processing step.

POET melds several well-suited practices of previous modern AE schemes: the masking process and the middle ECB layer follow the secure XEX approach [46], which provides security against chosen-plaintext and chosen-ciphertext adversaries. The header-processing procedure bases on PMAC1 [12] (see Figure 5.2), which is fast, fully parallelizable, and provably secure. Finally, the processing of the final message block and the tag-generation procedure of POET adopts the length-preserving and provably secure tag splitting from McOE [23].

In the remainder of this chapter, we first provide a formal definition of POET. Next, we describe the individual steps of the **key generation**, **header** and **message processing**, **tag generation** and **verification**, respectively. Prior, we define four auxiliary functions:

- $\mathbf{MSB}_b(X)$ returns the b most significant bits of X.
- $\mathbf{LSB}_b(X)$ returns the b least significant bits of X.
- $\mathbf{Split}_b(X)$ returns a tuple (X^α, X^β) where X^α contains the b most significant bits of X and X^β the $|X| - b$ least significant bits of X. Hence, $X^\alpha \mid\mid X^\beta = X$.
- $\mathbf{SplitInto}_b(X)$ defines the unique splitting of the given string X into substrings X_1, X_2, etc. such that $X = X_1 \mid\mid X_2 \mid\mid \ldots \mid\mid X_m$ and $|X_1| = |X_2| = \ldots = |X_{m-1}| = b$ and $X_m \leq b$.

5.1. Definition of POET

Definition 5.1 (POET). *Let $m, n, k \geq 1$ be fixed. Let $\ell_t \in \{0, \ldots, 128\}$ and $\ell_s \in \{0, \ldots, 2^{62} - 1\}$. Let POET $= (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an on-line AEAD scheme as defined in Definition 4.9, $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a block cipher and $F : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a family of keyed ϵ-AXU hash functions. Furthermore, let H be the header (including the public message number N appended to its end), M the message, T the authentication tag, and C the ciphertext, with $H, M, C \in \{0,1\}^*$ and $T \in \{0,1\}^n$. Then, $\mathcal{E}$ is given by procedure **EncryptAndAuthenticate**, $\mathcal{D}$ by procedure **DecryptAndVerify**, and $\mathcal{K}$ by procedure **GenerateKeys**, as shown in Algorithms 2 and 3, respectively.*

Support for Intermediate Tags. POET provides two parameters to adjust the size and frequency of intermediate tags:

- $\ell_t \in \{0, \ldots, 128\}$ denotes the number of bits of each intermediate tag. For simplicity, we fix the size of each intermediate tag for our recommendations to a full single block (n bits) throughout this work.
- $\ell_s \in \{0, \ldots, 2^{62} - 1\}$ represents the number of n-bit blocks between each pair of subsequent intermediate tags.

Key Generation. POET requires in total three pairwise independent k-bit keys: a key K for the block cipher, a masking key L for processing the header, and a key K_F for the keyed family of hash functions F.

The key generation follows the idea from [32]. The user supplies a k-bit secret key SK. The further keys are then generated by encrypting distinct constants $const_0$, $const_1$, $const_2$ under E_{SK}. For simplicity, we set $const_i = i$. Under the assumption that E is a secure pseudo-random permutation, we can ensure to obtain pairwise independent keys for the block cipher invocation and the masking.

Header Processing. The header H denotes the associated data of a message, and an n-bit nonce N appended to its end. Hence, one can also interpret the entire header as a nonce. We always apply the common 10*-padding, i.e., we append a single '1'-bit to the header followed by as many '0'-bits as necessary such that the length of the padded header becomes a multiple of n. Due to the encoding of ℓ_s and ℓ_t, the header is never empty. This is a mandatory requirement for POET since it generates an intermediate value τ that is used later to generate the authentication tag.

POET processes the header in a similar fashion as PMAC1 [12]. It differs from PMAC1 in the way that we replace the multiplication by $3L$ or $5L$ for the mask of the final block by an additional call

Algorithm 2 The procedures **EncryptAndAuthenticate** and **DecryptAndVerify** of POET with support for intermediate tags when $\ell_t = n$.

```
EncryptAndAuthenticate($\ell_s, \ell_t, H, M$)
101: $\widetilde{H} \leftarrow (\langle \ell_s \rangle_{n/2} \,||\, \langle \ell_t \rangle_{n/2} \,||\, H)$
102: $\tau \leftarrow$ ProcessHeader($\widetilde{H}$)
103: $\mu \leftarrow \lceil |M|/(\ell_s \cdot n) \rceil$
104: $(\widetilde{M}^1, \ldots, \widetilde{M}^\mu) \leftarrow$ Encode($\ell_s, \ell_t, M$)
105: $\widetilde{M} \leftarrow (\widetilde{M}^1 \,||\, \ldots \,||\, \widetilde{M}^\mu)$
106: $\tilde{m} \leftarrow \lceil |\widetilde{M}|/n \rceil$
107: $(C, X_{\tilde{m}}, Y_{\tilde{m}}) \leftarrow$ Encrypt($\widetilde{M}, \tau$)
108: $(C_{\tilde{m}}, T^\alpha) \leftarrow$ Split$_{|M_{\tilde{m}}|}(C_{\tilde{m}})$
109: $T^\beta \leftarrow$ GenerateTag($\tau, X_{\tilde{m}}, Y_{\tilde{m}}$)
110: $T \leftarrow T^\alpha \,||\, T^\beta$
111: return $(C_1 \,||\, \ldots \,||\, C_{\tilde{m}}, T)$
```

```
DecryptAndVerify($\ell_s, \ell_t, H, C, T$)
401: $\widetilde{H} \leftarrow (\langle \ell_s \rangle_{n/2} \,||\, \langle \ell_t \rangle_{n/2} \,||\, H)$
402: $\tau \leftarrow$ ProcessHeader($\widetilde{H}$)
403: $\tilde{m} \leftarrow \lceil |C|/n \rceil$
404: $\mu \leftarrow \lceil |C|/(\ell_s \cdot n + \ell_t) \rceil$
405: $(\widetilde{M}, X_{\tilde{m}}, Y_{\tilde{m}}) \leftarrow$ Decrypt($C, T, \tau$)
406: $(M^1, \ldots, M^j) \leftarrow$ Decode($\ell_s, \ell_t, \widetilde{M}$)
407: if $j < \mu$ then
408:     return $(M^1 \,||\, \ldots \,||\, M^j)$
409: $M \leftarrow (M^1 \,||\, \ldots \,||\, M^j)$
410: $(M_1, \ldots, M_m) \leftarrow$ SplitInto$_{\ell_s \cdot n}(M)$
411: $(M_m, \tau') \leftarrow$ Split$_{|C_m|}(M_m)$
412: if VerifyTag($T, X_m, Y_m, \tau, \tau'$) then
413:     return $(M_1 \,||\, \ldots \,||\, M_m)$
414: return $(M^1 \,||\, \ldots \,||\, M^{\mu-1})$
```

```
Encode($\ell_s, \ell_t, M$)
201: $\mu \leftarrow \lceil |M|/(\ell_s \cdot n) \rceil$
202: $(M^1, \ldots, M^\mu) \leftarrow$ SplitInto$_{\ell_s \cdot n}(M)$
203: for $i \leftarrow 1, \ldots, \mu - 1$ do
204:     $\widetilde{M}^i \leftarrow$ EncodePart($\ell_s, \ell_t, M^i$)
205: $\widetilde{M}^\mu \leftarrow M^\mu$
206: return $(\widetilde{M}^1 \,||\, \ldots \,||\, \widetilde{M}^\mu)$
```

```
Decode($\ell_s, \ell_t, \widetilde{M}$)
501: $\mu \leftarrow \lceil |\widetilde{M}|/(\ell_s \cdot n + \ell_t) \rceil$
502: $(\widetilde{M}^1, \ldots, \widetilde{M}^\mu) \leftarrow$ SplitInto$_{\ell_s \cdot n + \ell_t}(\widetilde{M})$
503: for $i \leftarrow 1, \ldots, \mu - 1$ do
504:     $M^i \leftarrow$ DecodePart($\ell_s, \ell_t, \widetilde{M}^i$)
505:     if $M^i = \perp$ then
506:         return $(M^1 \,||\, \ldots \,||\, M^{i-1})$
507: return $(M^1 \,||\, \ldots \,||\, M^\mu)$
```

```
EncodePart($\ell_s, \ell_t, M^i$)
301: return $(M^i \,||\, 0^n)$
```

```
DecodePart($\ell_s, \ell_t, \widetilde{M}^i$)
601: $(M^i \,||\, R) \leftarrow$ Split$_{\ell_s \cdot n}(\widetilde{M}^i)$
602: if $R = 0^n$ then
603:     return $M^i$;
604: return $\perp$
```

to E, which can help to reduce the area in hardware. Each block is masked by XORing a distinct multiple of L. Note that all multiplications are Galois-Field Multiplications in $GF(2^{128})$ using the irreducible polynomial $\mathtt{x}^{128} + \mathtt{x}^7 + \mathtt{x}^2 + \mathtt{x} + \mathtt{1}$. Each masked header block is used as input to E and all outputs are XORed together. The XOR sum is then encrypted again by E to generate a value τ. The procedure **ProcessHeader** is shown in Algorithm 3. Note that one can parallelize almost the entire header-processing step.

Encryption/Decryption. The workflow of the message encryption is shown in Algorithm 4 and Figure 5.1. For each message block M_i, for $1 \leq i \leq m - 1$, the following process is applied: update the top- and bottom-row chaining values X_{i-1} and Y_{i-1} by applying the ϵ-AXU functions $F(K_F, X_{i-1})$ and $F(K_F, Y_{i-1})$, respectively. Then, XOR the output of $F(K_F, X_{i-1})$ with the current message block M_i to derive X_i. The value X_i is then used twice; (1) as an input to the block cipher E and (2) as the new chaining value in the top row. The output $Y_i = E_K(X_i)$ is also used twice; (1) as the new chaining value for the bottom-row and (2) is XORed with the updated chaining value $F(K_F, Y_{i-1})$ to generate the current ciphertext block C_i. The decryption process is defined similarly in procedure **Decrypt**. For simplicity, we write $F_{K_F}(\cdot)$ instead of $F(K_F, \cdot)$ hereafter.

To process the final message block M_m, we first perform header and message processing. Then, POET encrypts the length of the message and XORs the result $S = E_K(|M|)$ to the final message block M_m. The result of $M_m \oplus S$ is then XORed with $F_{K_F}(X_{m-1})$ to produce X_m. The value

Algorithm 3 The procedures **GenerateKeys** and **ProcessHeader**.

```
GenerateKeys($SK$)
301: $K \leftarrow E_{SK}(const_0)$
302: $L \leftarrow E_{SK}(const_1)$
303: $K_F \leftarrow E_{SK}(const_2)$
304: return $(K, L, K_F)$
```

```
ProcessHeader($H$)
401: $H \leftarrow H \,||\, 10^*$
402: $h \leftarrow \lceil |H|/n \rceil$
403: $(H_1, \ldots, H_h) \leftarrow \mathbf{SplitInto}_n(H)$
404: $\Sigma \leftarrow 0^n$
405: for $i \leftarrow 1, \ldots, h$ do
406:     $\Sigma \leftarrow \Sigma \oplus E_K(H_i \oplus 2^{i-1}L)$
407: $\tau \leftarrow E_K(\Sigma)$
408: return $\tau$
```

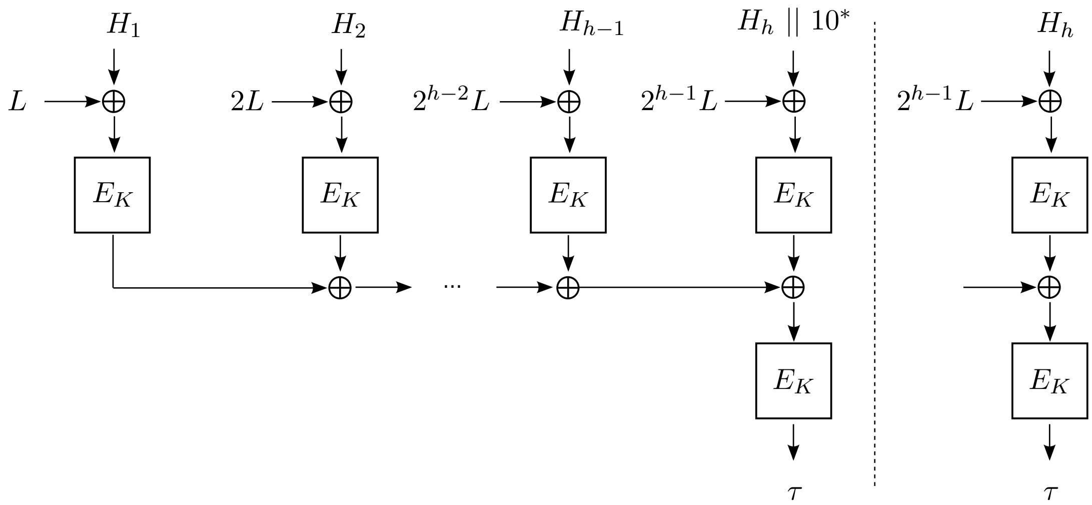


Figure 5.2.: The **ProcessHeader** procedure of POET. E denotes a block cipher and L a masking key that is updated for each block by doubling in Galois-Field $GF(2^{128})$. Processing of the final block differs depending on whether the final block is full (right) or not (left).

Algorithm 4 The procedures **Encrypt** and **Decrypt**.

```
Encrypt($M, \tau$)
501: $m \leftarrow \lceil |M|/n \rceil$, $X_0 \leftarrow \tau$, $Y_0 \leftarrow \tau \oplus 1$
502: $(M_1, \ldots, M_m) \leftarrow \mathbf{SplitInto}_n(M)$
503: for $i \leftarrow 1, \ldots, m-1$ do
504:     $X_i \leftarrow F_{K_F}(X_{i-1}) \oplus M_i$
505:     $Y_i \leftarrow E_K(X_i)$
506:     $C_i \leftarrow F_{K_F}(Y_{i-1}) \oplus Y_i$
507: $S \leftarrow E_K(|M|)$
508: $\tau^\alpha \leftarrow \mathbf{MSB}_{n-|M_m|}(\tau)$
509: $M_m^* \leftarrow (M_m \,||\, \tau^\alpha)$
510: $X_m \leftarrow F_{K_F}(X_{m-1}) \oplus M_m^* \oplus S$
511: $Y_m \leftarrow E_K(X_m)$
512: $C_m \leftarrow F_{K_F}(Y_{m-1}) \oplus Y_m \oplus S$
513: $C \leftarrow (C_1 \,||\, \ldots \,||\, C_m)$
514: return $(C, X_m, Y_m)$
```

```
Decrypt($C, T, \tau$)
601: $m \leftarrow \lceil |C|/n \rceil$, $X_0 \leftarrow \tau$, $Y_0 \leftarrow \tau \oplus 1$
602: $(C_1, \ldots, C_m) \leftarrow \mathbf{SplitInto}_n(C)$
603: for $i \leftarrow 1, \ldots, m-1$ do
604:     $Y_i \leftarrow F_{K_F}(Y_{i-1}) \oplus C_i$
605:     $X_i \leftarrow E_K^{-1}(Y_i)$
606:     $M_i \leftarrow F_{K_F}(X_{i-1}) \oplus X_i$
607: $S \leftarrow E_K(|C|)$
608: $T^\alpha \leftarrow \mathbf{LSB}_{n-|C_m|}(T)$
609: $C_m^* \leftarrow (C_m \,||\, T^\alpha)$
610: $Y_m \leftarrow F_{K_F}(Y_{m-1}) \oplus C_m^* \oplus S$
611: $X_m \leftarrow E_K^{-1}(Y_m)$
612: $M_m \leftarrow F_{K_F}(X_{m-1}) \oplus X_m \oplus S$
613: $M \leftarrow (M_1 \,||\, \ldots \,||\, M_m)$
614: return $(M, X_m, Y_m)$
```

X_m is again used twice; (1) as input to the block cipher call and (2) as chaining input to the tag-generation step. The output $Y_m = E_K(X_m)$ is also used twice; (1) as the new bottom-row chaining value for the tag generation and (2) it it is XORed with the updated chaining value $F_{K_F}(Y_{m-1})$. Thereupon, to produce the final ciphertext block, the result of the XOR operation is XORed again with the encrypted message length S.

For messages whose length is not a multiple of the block size, we employ a slightly more complicated procedure for the final block. To avoid overhead when transmitting the message, POET borrows the provably secure *tag-splitting* technique from McOE [22]. This means that messages are never padded; instead, the final message block M_m is filled up with the most significant bits of the intermediate tag τ:

$$M^*_m \leftarrow M_m \mid\mid \mathbf{MSB}_{n-|M_m|}(\tau),$$

where n denotes the block length. M^*_m is then encrypted as described above for the final message block. C^*_m is then split, where its $|M_{\ell_M}|$ most significant bits are used as the final bits of the ciphertext and the remaining bits as the $n - |M_m|$ most significant bits of the tag, T^α:

$$C_m \leftarrow \mathbf{MSB}_{|M_m|}(C^*_m), \quad T^\alpha \leftarrow \mathbf{LSB}_{n-|M_m|}(C^*_m).$$

The remaining bits of the tag are produced as shown in procedure **GenerateTag** in Algorithm 5.

Algorithm 5 The procedures **GenerateTag** and **VerifyTag**.

```
GenerateTag($\tau, X_m, Y_m$)
701: $X_{m+1} \leftarrow F_{K_F}(X_m) \oplus \tau$
702: $Y_{m+1} \leftarrow E_K(X_{m+1})$
703: $C^*_{m+1} \leftarrow F_{K_F}(Y_m) \oplus Y_{m+1} \oplus \tau$
704: $(T^\beta, Z) \leftarrow \mathbf{Split}_{|M_m|}(C^*_{m+1})$
705: return $T^\beta$

VerifyTag($T, X_m, Y_m, \tau, \tau'$)
801: $X_{m+1} \leftarrow F_{K_F}(X_m) \oplus \tau$
802: $Y_{m+1} \leftarrow E_K(X_{m+1})$
803: $C^*_{m+1} \leftarrow F_{K_F}(Y_m) \oplus Y_{m+1} \oplus \tau$
804: $(T', Z) \leftarrow \mathbf{Split}_{n-|\tau'|}(C^*_{m+1})$
805: $T^\beta \leftarrow \mathbf{LSB}_{n-|\tau'|}(T)$
806: $\tau^\alpha \leftarrow \mathbf{MSB}_{|\tau'|}(\tau)$
807: return $\tau^\alpha = \tau' \wedge T' = T^\beta$
```

Authentication/Verification. To generate or to verify the authentication tag, POET processes the intermediate tag τ similarly to a message block M_i (cf. Algorithm 5. The only difference is that τ is also XORed with the output of the cipher. When the length of the message is a multiple of n, the entire output C^*_{m+1} (cf. Line 703) is used as a tag, which is transmitted together with the ciphertext; for verification, C^*_{m+1} is compared to the tag which was transmitted with the ciphertext.

When the message length is not a multiple of n, we already obtained the first $n - |M_m|$ bits of the tag (T^α) from the encryption of the final message block (cf. Line 108 of Algorithm 2). The remaining $|M_m|$ bits of the tag (T^β) are taken from the $|M_m|$ most significant bits of C^*_{m+1} (cf. Line 704 of Algorithm 5); the rest of C^*_{m+1} is discarded. The concatenation of $T^\alpha \mid\mid T^\beta$ gives the authentication tag.

The verification consists of two steps: first, the $n - |M_m|$ least significant bits of M^*_m are compared with the $n - |M_m|$ most significant bits of τ. Thereupon, the $|M_m|$ most significant bits of C^*_{m+1} are compared to the $|M_m|$ least significant bits of T. If both checks are valid, the decrypted ciphertext is output; otherwise, the decryption fails (cf. lines 412 to 414 of Algorithm 2).

5.2. Instantiations for the Family of Hash Functions

We highly recommend to instantiate POET with AES-128 as a block cipher. For the ϵ-AXU families of hash functions F, we propose two instantiations POET-AES10-AES4 and POET-AES10-AES10.

POET-AES10-AES4. When trying to minimize the implementation footprint, it may be desirable to have an encryption scheme based on a single primitive. Furthermore, as mentioned before, maximizing the throughput is often critical. Therefore, POET with four-round AES as family of keyed hash functions may be an excellent choice for restricted devices and/or devices with integrated AES-NI. In particular, the key schedule of the keyed hash function needs to be called only once for a given key. The drawback of this solution would be a slightly lower number of message blocks that can be processed under the same key. Note that by four-round AES (AES4, hereafter) we mean a version of the AES-128 reduced to the first four rounds with pre- and post-whitening, including the MixColumns operation in the final round.

Keliher and Sui showed in [34] that the maximum expected differential probability (MEDP) of AES4 (under the assumption of independent round keys) is at most about $2^{-113.088}$, from which follows that AES4 is an ϵ-AXU family of hash functions with $\epsilon \leq 2^{-113}$.

POET-AES10-AES10. As a conservative variant, we propose to use full AES-128 as a family of hash functions. Under the common PRF assumption – where we assume that AES is indistinguishable from a random 128-bit permutation, this construction yields $\epsilon \approx 2^{-128}$.

5.3. Recommended Parameter Sets

We have two recommended versions of POET whose parameters are summarized below.

Primary Recommendation: POET-AES10-AES4

- Block cipher: AES-128.
- Hash function: AES4.
- Sizes: 128-bit key, 128-bit nonce, 128-bit state, 128-bit tag.
- Restricted to: $\ll 2^{56}$ 128-bit blocks per key

Secondary Recommendation: POET-AES10-AES10

- Block cipher: AES-128.
- Hash function: AES-128.
- Sizes: 128-bit key, 128-bit nonce, 128-bit state, 128-bit tag.
- Restricted to: $\ll 2^{64}$ 128-bit blocks per key

We do not priorize intermediate tags. Each recommended version can be used with intermediate tags, where we recommend ($\ell_s = 128, \ell_t = 128$) or without intermediate tags ($\ell_s = 0, \ell_t = 0$). Though, application-specific constraints may justify other values when intermediate tags are used.

5.4. Specification of POE

The encryption and decryption functions of POET– when considered without processing associated data and authentication – define a self-contained family of fast and secure on-line ciphers, called POE. While we concentrate on authenticated encryption in this work, we can profit from considering the encryption process in an isolated fashion for our later security discussion of POET. Therefore, we briefly define the POE family of on-line ciphers. Note that POE is defined only for messages whose length is a multiple of n. The key-generation for POE is similar to POET (Algorithm 3), except the steps in lines 302 and 303 are neglected since POE considers neither associated data nor authentication.

Definition 5.2 (POE). *Let $k, n \geq 1$ be two integers, $E : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a block cipher, and $F : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a family of keyed ϵ-AXU hash functions. Further, let $K, K_F \in \{0,1\}^k$ denote pairwise independent keys. Then, the encryption of POE and its inverse are defined by the procedures* ***Encrypt*** *and* ***Decrypt*** *as shown in Algorithm 6.*

Algorithm 6 The procedures **Encrypt** and **Decrypt** for POE.

```
Encrypt($M$)
101: $m \leftarrow |M|/n,\ X_0 \leftarrow 1, Y_0 \leftarrow 2$
102: for $i \leftarrow 1, \ldots, m$ do
103:     $X_i \leftarrow F_{K_F}(X_{i-1}) \oplus M_i$
104:     $Y_i \leftarrow E_K(X_i)$
105:     $C_i \leftarrow F_{K_F}(Y_{i-1}) \oplus Y_i$
106: return $C \leftarrow (C_1 \,||\, \ldots \,||\, C_m)$
```

```
Decrypt($C$)
201: $m \leftarrow |C|/n,\ X_0 \leftarrow 1, Y_0 \leftarrow 2$
202: for $i \leftarrow 1, \ldots, m$ do
203:     $Y_i \leftarrow F_{K_F}(Y_{i-1}) \oplus C_i$
204:     $X_i \leftarrow E_K^{-1}(Y_i)$
205:     $M_i \leftarrow F_{K_F}(X_{i-1}) \oplus X_i$
206: return $M \leftarrow (M_1 \,||\, \ldots \,||\, M_m)$
```

Chapter 6

Security Notions

This section recalls the security notions which consider the security of AE schemes. The literature provides various notions and relations for deterministic [48, 49] and nonce-based AE schemes [4, 6, 7, 33, 45, 47]. We consider the common CCA3 notion by Rogaway and Shrimpton [48] and recall the related IND-CPA and INT-CTXT notions for privacy and integrity which are covered by CCA3. Thereupon, we point out the differences between on-line and off-line encryption by defining the OCCA3 notion for the security of on-line AE schemes in general and the OCPA and OCCA notions for privacy in particular. To investigate the security of on-line AE schemes with intermediate tags, we develop similar notions OCPA-IT and INT-CTXT-IT for privacy and integrity. We follow the approach from [22] and provide a game for each notion that illustrates the interaction of the respective adversaries with their oracles.

6.1. General Security Notions for AE Schemes

Definition 6.1 (CCA3 Security). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an AE scheme as defined in Definition 4.9. Let $\mathcal{A}$ be a computationally bounded adversary. Then, the CCA3 advantage of $\mathcal{A}$ is defined as*

$$\mathbf{Adv}_{\Pi}^{\mathsf{CCA3}}(\mathcal{A}) = \left|\Pr\left[\mathcal{A}^{\mathcal{E}_K(\cdot,\cdot),\mathcal{D}_K(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot),\perp(\cdot,\cdot,\cdot)} \Rightarrow 1\right]\right|, \tag{6.1}$$

where the probabilities are taken over $K \twoheadleftarrow \mathcal{K}$. Further, we define $\mathbf{Adv}_{\Pi}^{\mathsf{CCA3}}(q, \ell, t)$ as the maximum advantage over all CCA3 adversaries $\mathcal{A}$ on Π that run in time at most t, and make at most q queries of total length ℓ to the available oracles.

The CCA3 notion states that $\mathcal{A}$ has access to a challenger that provides $\mathcal{A}$ with two oracles $\mathcal{O}_1$ for encryption and $\mathcal{O}_2$ for decryption. At the beginning, the challenger tosses a fair coin; depending on the result of the coin toss, it uses either the real encryption $\mathcal{E}_K(\cdot,\cdot)$ and decryption $\mathcal{D}_K(\cdot,\cdot,\cdot)$ functions or a random function $\$(\cdot,\cdot)$ for encryption and a $\perp(\cdot,\cdot,\cdot)$ function for decryption. $\$$ outputs random strings of the expected length. Considering the authenticated ciphertext to include the authentication tag, then $|\$(H,M)| = |\mathcal{E}_K(H,M)|$ for all keys $K \in \mathcal{K}$, headers $H \in \mathcal{H}$, and messages $M \in \mathcal{M}$. $\perp$ outputs the invalid symbol $\perp$ for every input. Wlog., we assume that $\mathcal{A}$ never asks a query to which it already knows the answer. The goal of $\mathcal{A}$ is to determine the result of the coin toss, i.e., to distinguish between the real encryptions with Π and random.

Definition 6.2 (IND-CPA Security). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an AE scheme as defined in Definition 4.9. Let $\mathcal{A}$ be a computationally bounded adversary. Then, the IND-CPA advantage of $\mathcal{A}$ is defined as*

$$\mathbf{Adv}_{\Pi}^{\textsf{IND-CPA}}(\mathcal{A}) = \left|\Pr\left[\mathcal{A}^{\mathcal{E}_K(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot)} \Rightarrow 1\right]\right|,$$

where the probabilities are taken over $K \twoheadleftarrow \mathcal{K}$ and the random coins of $\mathcal{A}$. Further, we define $\mathbf{Adv}_{\Pi}^{\textsf{IND-CPA}}(q, \ell, t)$ as the maximum advantage over all IND-CPA adversaries $\mathcal{A}$ on Π that run in time at most t, and make at most q queries of total length ℓ to the available oracles.

The INT-CTXT notion is the standard notion for the integrity of AE schemes. Though, we consider instead the INT-RUP notion that was introduced by Andreeva et al. [4] to study the integrity security of separated AE schemes, this means to model settings where the adversary sees also the decryption of invalid ciphertexts. Andreeva et al. [4] showed that INT-RUP security implies INT-CTXT security. Hence, it suffices to consider the INT-RUP advantage of an AE scheme to also obtain an upper bound for its INT-CTXT security.

Definition 6.3 (INT-RUP-Advantage). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{V})$ be a separated AE scheme as defined in Definition 4.10), and $\mathcal{A}$ a computationally bounded adversary with access to three oracles $\mathcal{O}_1, \mathcal{O}_2$, and $\mathcal{O}_3$ such that $\mathcal{A}$ never queries $\mathcal{O}_1 \hookrightarrow \mathcal{O}_3$. Then, the INT-RUP advantage of $\mathcal{A}$ with respect to Π is defined as*

$$\mathbf{Adv}_{\Pi}^{\textsf{INT-RUP}}(\mathcal{A}) := \Pr\left[\mathcal{A}^{\mathcal{E}_K, \mathcal{D}_K, \mathcal{V}_K} \textit{ forges}\right],$$

where the probability is taken over $K \twoheadleftarrow \mathcal{K}$. "Forges" means that $\mathcal{V}_K$ returns ***true*** *for a query of $\mathcal{A}$. We define $\mathbf{Adv}_{\Pi}^{\textsf{INT-RUP}}(q, \ell, t)$ as the maximum advantage over all INT-RUP adversaries $\mathcal{A}$ on Π that run in time at most t, and make at most q queries of total length ℓ to the available oracles.*

Let $G_{\textsf{INT-RUP}}$ be the INT-RUP game as shown in Figure 6.1. Then, the advantage of $\mathcal{A}$ in breaking Π in the INT-RUP setting can also be written as

$$\mathbf{Adv}_{\Pi}^{\textsf{INT-RUP}}(\mathcal{A}) \leq \Pr\left[\mathcal{A}^{G_{\textsf{INT-RUP}}} \Rightarrow 1\right].$$

```
Initialize()
   $K \twoheadleftarrow \mathcal{K}; \mathcal{Q} \leftarrow \emptyset$;
   win ← false;

Finalize()
   return win;

Encrypt($H, M$)
   $(C, T) \leftarrow \mathcal{E}_K(H, M)$;
   $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(H, C, T)\}$;
   return $(C, T)$;

Decrypt($H, C, T$)
   return $\mathcal{D}_K(H, C, T)$;

Verify($H, C, T$)
   $b \leftarrow \mathcal{V}_K(H, C, T)$
   if $b$ = true
      and $((H, C, T) \notin \mathcal{Q})$ then
      win ← true;
   end if
   return $b$;
```

Figure 6.1.: The INT-RUP game $G_{\textsf{INT-RUP}}$ for a separated authenticated encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{V})$.

Bellare and Namprempre showed in [6] that the CCA3 advantage of an adversary $\mathcal{A}$ on Π can be upper bounded by the sum of the maximal advantage of a chosen-plaintext adversary on the privacy and the maximal advantage of an adversary on the integrity of Π. Fleischmann et al. [22] generalized this relation, rewriting Equation (6.4) to

$$\Big| \Pr\left[\mathcal{A}^{\mathcal{E}_K(\cdot,\cdot),\mathcal{D}_K(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathcal{E}_K(\cdot,\cdot),\perp(\cdot,\cdot,\cdot)} \Rightarrow 1\right] \tag{6.2}$$

$$+ \Pr\left[\mathcal{A}^{\mathcal{E}_K(\cdot,\cdot),\perp(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot),\perp(\cdot,\cdot,\cdot)} \Rightarrow 1\right] \Big|, \tag{6.3}$$

where the probabilities are over $K \twoheadleftarrow \mathcal{K}$. Equation (6.2) refers to the INT-CTXT advantage and Equation (6.3) to the IND-CPA advantage of $\mathcal{A}$ on Π.

Theorem 6.4 (Theorem 1 in [22]). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an AE scheme as defined in Definition 4.9. Then, the CCA3 advantage over all adversaries $\mathcal{A}$ on Π that run in time at most t, ask at most q queries of a total length of at most ℓ blocks to the available oracles can be upper bounded by*

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

6.2. Security Notions for On-Line AE Schemes

Concerning privacy, we recall briefly the OPERMCCA notion for on-line ciphers by Bellare et al. [5].

Definition 6.5 (OPERMCCA Security for On-Line Ciphers). *Let $\Gamma : \mathcal{K} \times (\{0,1\}^n)^* \to (\{0,1\}^n)^*$ be an on-line cipher as defined in Definition 4.6. Let $\mathcal{A}$ be a computationally bounded adversary. Then, the OPERMCCA advantage of $\mathcal{A}$ with respect to Π is defined as*

$$\mathbf{Adv}_{\Gamma}^{\textsf{OPERMCCA}}(\mathcal{A}) = \left| \Pr\left[\mathcal{A}^{\Gamma_K(\cdot),\Gamma_K^{-1}(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{P(\cdot),P^{-1}(\cdot)} \Rightarrow 1\right] \right|, \tag{6.4}$$

where the probabilities are taken over $K \twoheadleftarrow \mathcal{K}$, $P \twoheadleftarrow \textsf{OPerm}_n$. Further, we define $\mathbf{Adv}_{\Pi}^{\textsf{OPERMCCA}}$ (q,ℓ,t) as the maximum advantage over all OPERMCCA adversaries $\mathcal{A}$ on Γ that run in time at most t, and make at most q queries of total length ℓ to the available oracles.

We have to adapt the OPERMCCA notion above so that we can use it to bound the security of on-line AE schemes. An on-line permutation P is tweaked by all previous blocks M_1 through M_{i-1} when computing the output for M_i. When concerning the security of on-line AEAD schemes, we want the encryption oracle in the random case, \$, to behave like an on-line permutation whose tweak space includes the associated data. Moreover, it should treat the final block of a message differently than non-final blocks, which can be modeled by extending the tweak space of an on-line permutation by an additional parameter that indicates whether the current input represents a final or non-final block. For this purpose, we derive the notion of on-line permutations with extended domain in Definition 6.6.

Definition 6.6 (Domain-Extended On-Line Permutation). *Let $F_i : \{0,1\} \times \{0,1\}^* \times (\{0,1\}^n)^i \to \{0,1\}^n$ be a family of indexed n-bit permutations, i.e., for any fixed $g \in \{0,1\}$, fixed index $j \in (\{0,1\}^n)^{i-1}$, and fixed $h \in \{0,1\}^*$, it applies that $F_i(g,h,j,\cdot)$ is a permutation. We define an extended n-bit on-line permutation $P : \{0,1\} \times \{0,1\}^* \times (\{0,1\}^n)^m \to (\{0,1\}^n)^m$ as a composition of m extended indexed permutations $F_1 \cup F_2 \cup \cdots \cup F_m$. Given a fixed $h \in \{0,1\}^*$, an*

m-block message $M = (M_1, \ldots, M_m)$ is mapped to an m-block output $(C_1, \ldots, C_m)$ by

$$C_i = \begin{cases} F_i(0, H, M_1 \mid\mid \ldots \mid\mid M_{i-1}, M_i). & \textit{if } i \in [1, m-1]. \\ F_i(1, H, M_1 \mid\mid \ldots \mid\mid M_{m-1}, M_m), & \textit{if } i = m. \end{cases}$$

We define the set of all n-bit extended on-line permutations with domain $\{0,1\} \times \mathcal{H} \times (\{0,1\}^n)^*$ by $\mathsf{OPerm}_n^{\{0,1\}\times\mathcal{H}}$. Note that it is defined only for inputs whose length is a multiple of n.

Let $G \twoheadleftarrow \mathsf{Func}(\mathcal{H} \times \mathcal{M}, \mathcal{T})$ and $P \twoheadleftarrow \mathsf{OPerm}_n^{\{0,1\}\times\mathcal{H}}$. Let $\mathcal{M} = \mathcal{C} = (\{0,1\}^n)^*$. For the sake of brevity, we define the random function $\$_\mathrm{O} : \mathcal{H} \times \mathcal{M} \to \mathcal{C} \times \mathcal{T}$ to output $\$_\mathrm{O}(H, M) := (P(H, M), G(H, M))$. Analogously to CCA3, we define below the OCCA3 notion which concerns the security of on-line AE schemes against chosen-ciphertext adversaries.

Definition 6.7 (OCCA3 Security). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an on-line AE scheme, as defined in Definition 4.9. Let $\mathcal{A}$ be a computationally bounded adversary. Then, the OCCA3 advantage of $\mathcal{A}$ with respect to Π is defined as*

$$\mathbf{Adv}_{\Pi}^{\mathsf{OCCA3}}(\mathcal{A}) = \left| \Pr\left[\mathcal{A}^{\mathcal{E}_K(\cdot,\cdot), \mathcal{D}_K(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$_O(\cdot,\cdot), \perp(\cdot,\cdot,\cdot)} \Rightarrow 1\right] \right|, \tag{6.5}$$

where the probabilities are taken over $K \twoheadleftarrow \mathcal{K}$, $P \twoheadleftarrow \mathsf{OPerm}_n^{\{0,1\}\times\mathcal{H}}$ and $G \twoheadleftarrow \mathsf{Func}(\mathcal{H} \times \mathcal{M}, \mathcal{T})$. $\mathcal{H}$, $\mathcal{M}$, $\mathcal{C}$, and $\mathcal{T}$ denote header, message, ciphertext, and tag space, respectively. Further, we define $\mathbf{Adv}_{\Pi}^{\mathsf{OCCA3}}(q, \ell, t)$ as the maximum advantage over all OCCA3 adversaries $\mathcal{A}$ on Π that run in time at most t, and make at most q queries of total length ℓ to the available oracles.

As for CCA3, one could separate OCCA3 again into a notion for on-line privacy and the usual INT-CTXT notion for integrity, e.g., for the sake of proving the advantage for each notion separately. It is straight-forward to derive from IND-CPA an analogous notion of OCPA for on-line privacy against chosen-plaintext adversaries.

Definition 6.8 (OCPA Security). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an on-line AE scheme, as defined in Definition 4.10. Let $\mathcal{A}$ be a computationally bounded adversary. Then, we define the OCPA advantage of $\mathcal{A}$ with respect to Π as*

$$\mathbf{Adv}_{\Pi}^{\mathsf{OCPA}}(\mathcal{A}) = \left| \Pr\left[\mathcal{A}^{\mathcal{E}_K(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$_O(\cdot,\cdot)} \Rightarrow 1\right] \right|, \tag{6.6}$$

where the probabilities are taken over $K \twoheadleftarrow \mathcal{K}$, $P \twoheadleftarrow \mathsf{OPerm}_n^{\{0,1\}\times\mathcal{H}}$ and $G \twoheadleftarrow \mathsf{Func}(\mathcal{H} \times \mathcal{M}, \mathcal{T})$. $\mathcal{H}$, $\mathcal{M}$, $\mathcal{C}$, and $\mathcal{T}$ denote header, message, ciphertext, and tag space, respectively. Further, we define $\mathbf{Adv}_{\Pi}^{\mathsf{OCPA}}(q, \ell, t)$ as the maximum advantage over all OCPA adversaries $\mathcal{A}$ on Π that run in time at most t, and make at most q queries of total length ℓ to the available oracles.

Though, we are interested in investigating the privacy security of POET also against chosen-ciphertext adversaries. For this purpose, we consider a somewhat stronger notion than OCPA, which we call OCCA, which considers also decryption misuse. This means, we consider for the latter a separated on-line AE scheme for which the adversary sees also the output for invalid decryptions. We define the decryption oracle in the random world $\$_\mathrm{O}^{-1} : \mathcal{H} \times \mathcal{C} \times \mathcal{T} \to \mathcal{M}$ to output $\$_\mathrm{O}^{-1}(H, C, T) := P^{-1}(H, C)$ for all $H \in \mathcal{H}$, $C \in \mathcal{C}$, and $T \in \mathcal{T}$. Figure 6.2 shows the OCPA and OCCA games.

Definition 6.9 (OCCA Security). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{V})$ be a separated on-line AE scheme, as defined in Definition 4.10. Let $\mathcal{A}$ be a computationally bounded adversary. Then, we define the OCCA advantage of $\mathcal{A}$ with respect to Π as*

$$\mathbf{Adv}_{\Pi}^{\mathsf{OCCA}}(\mathcal{A}) = \left| \Pr\left[\mathcal{A}^{\mathcal{E}_K(\cdot,\cdot),\mathcal{D}_K(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$_O(\cdot,\cdot),\$_O^{-1}(\cdot,\cdot,\cdot)} \Rightarrow 1\right] \right|, \tag{6.7}$$

where the probabilities are taken over $K \twoheadleftarrow \mathcal{K}$, $P \twoheadleftarrow \mathsf{OPerm}_n^{\{0,1\}\times\mathcal{H}}$ and $G \twoheadleftarrow \mathsf{Func}(\mathcal{H} \times \mathcal{M}, \mathcal{T})$. $\mathcal{H}$, $\mathcal{M}$, $\mathcal{C}$, and $\mathcal{T}$ denote header, message, ciphertext, and tag space, respectively. Further, we define $\mathbf{Adv}_{\Pi}^{\mathsf{OCCA}}(q, \ell, t)$ as the maximum advantage over all OCCA adversaries $\mathcal{A}$ on Π that run in time at most t, and make at most q queries of total length ℓ to the available oracles.

Games G_{OCPA} and $\boxed{G_{\mathsf{OCCA}}}$

```
Initialize()
    $b \twoheadleftarrow \{0,1\}$; $\mathcal{Q} \leftarrow \emptyset$;
    if $b = 1$ then
        $K \twoheadleftarrow \mathcal{K}$;
    else
        $P \twoheadleftarrow \mathsf{OPerm}_n^{\{0,1\}\times\mathcal{H}}$;
        $G \twoheadleftarrow \mathsf{Func}(\mathcal{H} \times \mathcal{M}, \mathcal{T})$;
    end if

Finalize($b'$)
    return $b = b'$;
```

```
Encrypt($H, M$)
    if $b = 1$ then
        $(C, T) \leftarrow \mathcal{E}_K(H, M)$;
    else
        $C \leftarrow P(H, M)$;
        $T \leftarrow G(H, M)$;
    end if
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(H, C)\}$;
    return $(C, T)$;
```

```
Decrypt($H, C, T$)
    if $(H, C) \in \mathcal{Q}$ then
        return $\perp$;
    end if
    if $b = 1$ then
        return $\mathcal{D}_K(H, C, T)$;
    else
        return $P^{-1}(H, C)$;
    end if
```

Figure 6.2.: The game G_{OCPA} for an on-line authenticated encryption scheme (with associated data) $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ and the game G_{OCCA} for a separated on-line authenticated encryption scheme (with associated data) $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{V})$. The latter game contains the code in boxes while the former does not. $\mathcal{H}$, $\mathcal{M}$, and $\mathcal{T}$ denote header, message, and tag space, respectively.

Remark. In [6], Bellare and Namprempre showed that IND-CCA security implies non-malleable chosen-ciphertext (NM-CCA) security. Hence, OCCA security implies *weak* non-malleability, which means that an adversary that manipulates the i-th ciphertext block cannot distinguish the $(i+1)$-th, $(i+2)$-th, ... ciphertext blocks from random. Therefore, an OCCA-secure AE scheme provides weak non-malleability for nonce-ignoring and non-malleability for nonce-respecting adversaries. Note that an OCPA adversary $\mathcal{A}$ on a separated on-line AE scheme Π can always be used by an OCCA adversary $\mathcal{A}'$ on Π with at least the advantage of $\mathcal{A}$. Hence, an upper bound of the OCCA advantage of Π is also an upper bound of the OCPA advantage of Π.

Security Notions for On-Line AEAD Schemes with Intermediate Tags. On-line AE schemes with intermediate tags allow to release the first i verified parts before processing parts $i+1$, $i+2$, etc. We say that an on-line AE scheme with intermediate tags $\widetilde{\Pi} = (\mathcal{K}, \widetilde{\mathcal{E}}, \widetilde{\mathcal{D}})$ provides privacy if there exists no adversary which has non-negligible advantage in distinguishing the outputs from that of a random function which maintains the block-wise on-line behavior and which mimics the expected expansion of each part. We say that $\widetilde{\Pi}$ provides integrity if there exists no adversary which can forge any intermediate or final authentication tag for a non-previously seen sequence of parts with non-negligible advantage.

For concreteness, we define two notions OCPA-IT and INT-CTXT-IT for privacy against chosen-plaintext attacks and integrity, respectively. Since we consider the security depending only from that of Π and not from the used encoding, we define that the real and the ideal encryption oracle,

$\widetilde{\$}_{\mathrm{O}}$, use the same family of ℓ_t-expanding functions $\mathbf{EncodePart}_{\ell_s,\ell_t} : \{0,1\}^{\ell_s \cdot n} \to \{0,1\}^{\ell_s \cdot n + \ell_t}$ to expand non-final message parts. The game $G_{\mathsf{OCPA\text{-}IT}}$ is shown in Figure 6.3.

Definition 6.10 (OCPA-IT Security). *Let $\widetilde{\Pi} = (\mathcal{K}, \widetilde{\mathcal{E}}, \widetilde{\mathcal{D}})$ be an on-line AE scheme with intermediate tags, as given in Definition 4.11. Let further $n, x \geq 1$ be fixed and $\boldsymbol{EncodePart}_{\ell_s,\ell_t} : \{0,1\}^{\ell_s \cdot n} \to \{0,1\}^{\ell_s \cdot n + \ell_t}$ be a family of ℓ_t-expanding functions used in $\widetilde{\mathcal{E}}$ and let $\widetilde{\$}_O$ be defined as above. Let $\mathcal{A}$ be a computationally bounded adversary. Then, the OCPA-IT advantage of $\mathcal{A}$ with respect to $\widetilde{\Pi}$ is defined as*

$$\mathbf{Adv}_{\widetilde{\Pi}}^{\mathsf{OCPA\text{-}IT}}(\mathcal{A}) = \left| \Pr\left[\mathcal{A}^{\widetilde{\mathcal{E}}_K(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\widetilde{\$}_O(\cdot,\cdot)} \Rightarrow 1\right] \right|, \tag{6.8}$$

where the probabilities are taken over $K \twoheadleftarrow \mathcal{K}$, $P \twoheadleftarrow \mathsf{OPerm}_n^{\{0,1\} \times \mathcal{H}}$, and $G \twoheadleftarrow \mathsf{Func}(\mathcal{H} \times \mathcal{M}, \mathcal{T})$. $\mathcal{H}$, $\mathcal{M}$, $\mathcal{C}$, and $\mathcal{T}$ denote header, message, ciphertext, and tag space, respectively. Further, we define $\mathbf{Adv}_{\widetilde{\Pi}}^{\mathsf{OCPA\text{-}IT}}(q, \ell, t)$ as the maximum advantage over all OCPA-IT adversaries $\mathcal{A}$ on Π that run in time at most t, and make at most q queries of total length ℓ to the available oracles.

```
Game $G_{\mathsf{OCPA\text{-}IT}}$

Initialize()
    $b \twoheadleftarrow \{0,1\}$; $\mathcal{Q} \leftarrow \emptyset$;
    if $b = 1$ then
        $K \twoheadleftarrow \mathcal{K}$;
    else
        $P \twoheadleftarrow \mathsf{OPerm}_n^{\{0,1\} \times \mathcal{H}}$;
        $G \twoheadleftarrow \mathsf{Func}(\mathcal{H} \times \mathcal{M}, \mathcal{T})$;
    end if

Finalize($b'$)
    return $b = b'$;

Encrypt($\ell_s, \ell_t, H, M$)
    if $b = 1$ then
        $(C, T) \leftarrow \widetilde{\mathcal{E}}_K(\ell_s, \ell_t, H, M)$;
    else
        $\widetilde{H} \leftarrow (\langle \ell_s \rangle_x \| \langle \ell_t \rangle_x \| H)$;
        $\widetilde{M} \leftarrow \mathtt{Encode}_{\ell_s,\ell_t}(M)$;
        $C \leftarrow P(\widetilde{H}, \widetilde{M})$;
        $T \leftarrow G(\widetilde{H}, \widetilde{M})$;
    end if
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\ell_s, \ell_t, H, C, T)\}$;
    return $(C, T)$;
```

Figure 6.3.: The game $G_{\mathsf{OCPA\text{-}IT}}$ for an on-line AEAD scheme with intermediate tags $\widetilde{\Pi} = (\mathcal{K}, \widetilde{\mathcal{E}}, \widetilde{\mathcal{D}})$. $\ell_s, \ell_t, x \geq 1$ are fixed integers. The definition of the family of functions $\mathbf{Encode}_{\ell_s,\ell_t}$ is given in Algorithm 1 in Section 4.4.

Below, we define the INT-CTXT-IT notion to consider the integrity of on-line AE schemes with intermediate tags.

Definition 6.11 (INT-CTXT-IT Security). *Let $\widetilde{\Pi} = (\mathcal{K}, \widetilde{\mathcal{E}}, \widetilde{\mathcal{D}})$ be an on-line AE scheme with intermediate tags, as defined in Definition 4.11. Let $\mathcal{A}$ be a computationally bounded adversary. Let $G_{\mathsf{INT\text{-}CTXT\text{-}IT}}$ be the INT-CTXT-IT game as shown in Figure 6.4. Then, the advantage of $\mathcal{A}$ in breaking $\widetilde{\Pi}$ in the INT-CTXT-IT setting can be written as*

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

where the probability is taken over $K \twoheadleftarrow \mathcal{K}$. Further, we define $\mathbf{Adv}_{\widetilde{\Pi}}^{\mathsf{INT\text{-}CTXT\text{-}IT}}(q, \ell, t)$ as the maximum advantage over all INT-CTXT-IT adversaries $\mathcal{A}$ on $\widetilde{\Pi}$ that run in time at most t, and make at most q queries of total length of at most ℓ blocks to the available oracles.

Since the `Decrypt` procedure calls $\widetilde{\mathcal{D}}$ internally, the adversary gets to see only the longest sequence of decrypted parts $(M^1, \ldots, M^j)$ for which each encoded corresponding ciphertext part

$(C^1, \ldots, C^j)$ contained a valid intermediate tag or a valid final tag. So, $\mathcal{A}$ wins if `Decrypt` outputs some message M that is not part of a prefix of any previous query of $\mathcal{A}$. For this purpose, `Decrypt` compares the number of blocks in the longest common prefix of (ℓ_s, ℓ_t, H, M) with all previous queries (p) with the number of blocks in M (p'). So, if it holds that $p' > p$, then this implies that $\mathcal{A}$ could forge at least one intermediate (or final) tag.

```
Game $G_{\mathsf{INT\text{-}CTXT\text{-}IT}}$

Initialize()
    $K \twoheadleftarrow \mathcal{K}$; $\mathcal{Q} \leftarrow \emptyset$;
    win ← false;

Finalize()
    return win;

Encrypt($\ell_s, \ell_t, H, M$)
    $(C, T) \leftarrow \widetilde{\mathcal{E}}_K(\ell_s, \ell_t, H, M)$;
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\ell_s, \ell_t, H, C, T)\}$;
    return $(C, T)$;

Decrypt($\ell_s, \ell_t, H, C, T$)
    $M \leftarrow \widetilde{\mathcal{D}}_K(\ell_s, \ell_t, H, C, T)$;
    $p \leftarrow \mathsf{LLCP}_n(\mathcal{Q}, (\ell_s, \ell_t, H, C))$;
    $p' \leftarrow |M|/n$;
    if $p' > p$ then
        win ← true;
    end if;
    return $M$;
```

Figure 6.4.: The game $G_{\mathsf{INT\text{-}CTXT\text{-}IT}}$ for an on-line AEAD scheme with intermediate tags $\widetilde{\Pi} = (\mathcal{K}, \widetilde{\mathcal{E}}, \widetilde{\mathcal{D}})$. $\ell_s, \ell_t \geq 1$ are fixed integers.

Relations among Notions. We focus on online AE schemes for which the result of processing the header influences the process of all its message blocks. Given such an OCCA-secure scheme Π, the best that adversaries can see with non-negligible advantage is the longest common prefix to other messages. This statement holds for both nonce-respecting and nonce-ignoring adversaries. Though, it follows that Π also provides IND-CCA (and therefore, also IND-CPA) security when we consider nonce-respecting adversaries since the longest common prefix between all messages is always the empty string. Note that we always consider nonce-ignoring adversaries which are allowed to use a nonce multiple times similar to the security notions of integrity for authenticated encryption schemes in [22].

Chapter 7

Security Analysis

This chapter analyzes the security of POET concerning privacy and integrity. Since POET contains the on-line cipher POE, we start in Section 7.1 with the OPERMCCA analysis of POE. This allows us to derive our OPERMCCA arguments for POET in a straight-forward manner in Section 7.2. Thereupon, Section 7.3 considers the security of POET in the INT-RUP setting. These sections handle POET without intermediate tags. In the remaining two sections of this chapter, we derive slightly modified versions of our proofs for on-line privacy and integrity of POET when it is used with intermediate tags.

From the sum of the bounds in Theorem 7.2 and 7.3 follows the OCCA3 advantage of POET. From the fact that we consider also INT-RUP security and on-line privacy against chosen-ciphertext adversaries, follows that POET provides also robustness against decryption misuse.

Throughout the chapter, we denote by $\mathsf{POE}_{E,F}$ and $\mathsf{POE}^{-1}_{E^{-1},F}$ POE, instantiated with a given block cipher E and a family of ϵ-AXU hash functions F. Similarly, we denote by $\mathsf{POET}_{E,F,\ell_s,\ell_t}$ and $\mathsf{POET}^{-1}_{E^{-1},F,\ell_s,\ell_t}$ POET, instantiated with a given block cipher E and a family of ϵ-AXU hash functions F and studying the security using ℓ_s and ℓ_t as parameters of all queries by the considered adversaries.

7.1. OPERMCCA Security Analysis of POE

Theorem 7.1 (OPERMCCA-Security of POE). *Let $E \in \texttt{Block}(k,n)$ be a block cipher and $F : \{0,1\}^n \to \{0,1\}^n$ be an ϵ-AXU family of hash functions. Furthermore, let $F_i : \{0,1\}^{ni} \to \{0,1\}^n$ be an $(i \cdot \epsilon)$-AXU family of hash functions defined as follows:*

$$F_0 = F(1); \quad F_i(M) = F(F_{i-1}(M_1, \dots, M_{i-1}) \oplus M_i), \qquad i \in \mathbb{N}^+.$$

Then, it holds that

$$\mathbf{Adv}^{\mathsf{OPERMCCA}}_{\mathsf{POE}_{E,F}}(q,\ell,t) \leq \ell(\ell+1)\epsilon + \frac{\ell^2}{2^n - \ell} + \mathbf{Adv}^{\mathsf{IND\text{-}SPRP}}_{E,E^{-1}}(\ell, \mathcal{O}(t)).$$

Proof. Let $\mathcal{A}$ be an OPERMCCA adversary with access to an oracle $\mathcal{O}$, which responds either with real encryptions/decryptions using $\mathsf{POE}_{E,F}/\mathsf{POE}^{-1}_{E^{-1},F}$ or random encryptions/decryptions using P/P^{-1} as in Definition 6.10. In the beginning, the oracle tosses a fair coin to obtain a bit b.

Thereupon, $\mathcal{A}$ can query messages to $\mathcal{O}$. Depending on b, $\mathcal{A}$ obtains either "real" encryptions (or decryptions, respectively) for the messages it sends to $\mathcal{O}$ or "random" outputs. Hence, the task for $\mathcal{A}$ is to guess b. $\mathcal{A}$ asks at most q queries of a total length of at most ℓ blocks and stores each query together with the corresponding response from the oracle as tuples (M^i, C^i) in a query history $\mathcal{Q}$.

Wlog., we assume that $\mathcal{A}$ will not make queries to which it already knows the answer. It is easy to see that we can upper bound Equation (6.8) as (cf. [22], Section 4)

$$\left|\Pr\left[\mathcal{A}^{\mathsf{POE}_{E,F}(\cdot),\mathsf{POE}^{-1}_{E^{-1},F}(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathsf{POE}_{\pi,F}(\cdot),\mathsf{POE}^{-1}_{\pi^{-1},F}(\cdot)} \Rightarrow 1\right]\right| + \tag{7.1}$$

$$\left|\Pr\left[\mathcal{A}^{\mathsf{POE}_{\pi,F}(\cdot),\mathsf{POE}^{-1}_{\pi^{-1},F}(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{P(\cdot),P^{-1}(\cdot)} \Rightarrow 1\right]\right|, \tag{7.2}$$

where $\pi : \{0,1\}^n \to \{0,1\}^n$ denotes an n-bit random permutation that is chosen at random from the set of all n-bit random permutations, and π^{-1} denotes its inverse.

The difference in Equation (7.1) can be upper bounded by the maximal IND-SPRP advantage of an adversary which runs in time at most $\mathrm{O}(t)$, and asks at most ℓ queries to its oracles:

$$\mathbf{Adv}^{\mathsf{IND\text{-}SPRP}}_{E,E^{-1}}(\ell, \mathrm{O}(t)).$$

Proof Idea for the Remainder. It remains to study the difference in (7.2), which refers to the advantage of $\mathcal{A}$ to distinguish $\mathsf{POE}_{\pi,F}/\mathsf{POE}^{-1}_{\pi^{-1},F}$ from P/P^{-1}. From the structure of POE, we can identify two cases that can occur during the OPERMCCA game: (1) some collisions between internal values, or (2) no such collisions occur. We define the event COLL for the first case and $\neg$ COLL for the latter case. In the following we explain what we mean by internal collision. Using the law of total probability, we can reformulate Equation (7.2) to

$$\Pr[\mathsf{COLL}] \cdot \Pr[\mathsf{COLLWIN}] + \Pr[\neg\,\mathsf{COLL}] \cdot \Pr[\mathsf{NOCOLLWIN}], \tag{7.3}$$

where we define COLLWIN and NOCOLLWIN as the events that $\mathcal{A}$ wins in the COLL and NOCOLL-WIN case, respectively:

$$\Pr[\mathsf{COLLWIN}] = \left|\Pr\left[\mathcal{A}^{\mathsf{POE}_{\pi,F}(\cdot),\mathsf{POE}^{-1}_{\pi^{-1},F}(\cdot)} \Rightarrow 1 \,|\, \mathsf{COLL}\right] - \Pr\left[\mathcal{A}^{P(\cdot),P^{-1}(\cdot)} \Rightarrow 1\right]\right|, \text{ and} \tag{7.4}$$

$$\Pr[\mathsf{NOCOLLWIN}] = \left|\Pr\left[\mathcal{A}^{\mathsf{POE}_{\pi,F}(\cdot),\mathsf{POE}^{-1}_{\pi^{-1},F}(\cdot)} \Rightarrow 1 \,|\, \neg\,\mathsf{COLL}\right] - \Pr\left[\mathcal{A}^{P(\cdot),P^{-1}(\cdot)} \Rightarrow 1\right]\right|. \tag{7.5}$$

To simplify our proof, we can upper bound Equation (7.3) by

$$\Pr[\mathsf{COLL}] + \Pr[\mathsf{NOCOLLWIN}].$$

COLL. We concern two possible cases of internal collisions: (1) collisions between top-row chaining values X_i and X'_j, and (2) collisions between bottom-row chaining values Y_i and Y'_j. Clearly, we are not interested in collisions within a common prefix since the final goal of $\mathcal{A}$ is to distinguish POE from π.

Since π is chosen uniformly at random from the set of all n-bit on-line permutations, any "fresh" (i.e., not previously queried) input to $\pi(\cdot)$ or its inverse $\pi^{-1}(\cdot)$ produces a random output from the set of all non-previously output values from $\{0,1\}^n$. This implies for the internal values of POE that:

- For any fresh X_i, the result of $Y_i = \pi(X_i)$ is also random, and so are the resulting ciphertext outputs $C_i = Y_i \oplus F_{K_F}(Y_{i-1})$.
- For any fresh Y_i in decryption direction, the result of $X_i = \pi^{-1}(Y_i)$ is also random, and so are the resulting decrypted message blocks $M_i = X_i \oplus F_{K_F}(X_{i-1})$.

We define the event $\mathsf{COLL}^{\text{enc}}$ for an internal collision in the top-row chaining value, and the event $\mathsf{COLL}^{\text{dec}}$ for the bottom one. So, COLL represents the union event that either (or both) of $\mathsf{COLL}^{\text{enc}}$ or $\mathsf{COLL}^{\text{dec}}$ occurred. The maximum probability $\Pr[\mathsf{COLL}^{\text{enc}}]$ can be upper bounded by $\frac{\ell(\ell+1)\epsilon}{2}$. The proof follows from Lemma A.1. Due to the symmetric structure of POE, it holds that $\Pr[\mathsf{COLL}^{\text{dec}}] = \Pr[\mathsf{COLL}^{\text{enc}}]$.

NOCOLLWIN. The maximal probability for the event $\mathsf{NOCOLLWIN}$ can be upper bounded by $\frac{\ell^2}{2^n-\ell}$. The proof follows from Lemma A.2.
Our claim in Theorem 7.1 follows from summing up the individual terms. $\square$

7.2. OCCA Security Analysis of POET Without Intermediate Tags

Using our bound for the OCCA advantage of POE from the previous section, this section will go through a similar analysis to derive the OCCA advantage for POET without intermediate tags.

Theorem 7.2 (OCCA Security of POET When $\ell_s = 0$). *Let $E \in \texttt{Block}(k, n)$ be a block cipher and $F : \{0,1\}^n \to \{0,1\}^n$ be an ϵ-AXU family of hash functions. Furthermore, let $F_i : \{0,1\}^{ni} \to \{0,1\}^n$ be an $i\epsilon$-AXU family of hash functions defined as follows:*

$$F_0 = F(1); \quad F_i(M) = F(F_{i-1}(M_1, \ldots, M_{i-1}) \oplus M_i), \qquad i \in \mathbb{N}^+.$$

Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{V})$ be $\mathsf{POET}_{E,F,0,0}$ as defined in Definition 5.1 and the functions $\mathcal{D}$ and $\mathcal{V}$ as defined in Algorithm 7. Then, it holds that

$$\mathbf{Adv}_{\Pi}^{\mathsf{OCCA}}(q, \ell, t) \leq \frac{5.5(\ell + 2q)^2}{2^n} + (\ell + q)^2\epsilon + 2 \cdot \max\left\{q(\ell+q)\epsilon, \frac{q(\ell+q)}{2^n - q}\right\} + \frac{(\ell + 4q)^2}{2^n - (\ell + 4q)}$$
$$+ \mathbf{Adv}_{E,E^{-1}}^{\mathsf{IND\text{-}SPRP}}(\ell + 4q, O(t)).$$

Algorithm 7 The separated $\mathcal{D}_K(\ell_s, \ell_t, H, C, T)$ and $\mathcal{V}_K(\ell_s, \ell_t, H, C, T)$ functions of POET to simulate that the adversary sees also the decryptions of invalid ciphertexts.

```
D_K(ℓ_s, ℓ_t, H, C, T)
101: $\widetilde{H} \leftarrow (0^{n/2} \,||\, 0^{n/2} \,||\, H)$
102: $\tau \leftarrow \mathbf{ProcessHeader}(\widetilde{H})$
103: $m \leftarrow \lceil |C|/n \rceil$
104: $(M, X_m, Y_m) \leftarrow \mathbf{Decrypt}(C, T, \tau)$
105: $(M_m, \tau') \leftarrow \mathbf{Split}_{|C_m|}(M_m)$
106: return $M \leftarrow M_1 \,||\, \ldots \,||\, M_m$

V_K(ℓ_s, ℓ_t, H, C, T)
201: $\widetilde{H} \leftarrow (0^{n/2} \,||\, 0^{n/2} \,||\, H)$
202: $\tau \leftarrow \mathbf{ProcessHeader}(\widetilde{H})$
203: $m \leftarrow \lceil |C|/n \rceil$
204: $(M, X_m, Y_m) \leftarrow \mathbf{Decrypt}(C, T, \tau)$
205: $(M_m, \tau') \leftarrow \mathbf{Split}_{|C_m|}(M_m)$
206: return $\mathbf{VerifyTag}(T, X_m, Y_m, \tau, \tau')$
```

Proof. The proof follows the ideas of the OPERMCCA analysis of POE. Since we focus on $\ell_s = \ell_t = 0$, we can omit both parameters in this proof for brevity. Note, that the separated decrypt and verify functions omit the decoding since it would not alter the message and since there are no intermediate tags that had to be verified.

Similar to our OPERMCCA analysis of POE, we can rewrite Equation (6.7) as

$$\left| \Pr\left[\mathcal{A}^{\mathsf{POET}_{E,F,0,0}(\cdot,\cdot),\mathsf{POET}^{-1}_{E^{-1},F,0,0}(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathsf{POET}_{\pi,F,0,0}(\cdot,\cdot),\mathsf{POET}^{-1}_{\pi^{-1},F,0,0}(\cdot,\cdot,\cdot)} \Rightarrow 1\right] \right| + \quad (7.6)$$

$$\left| \Pr\left[\mathcal{A}^{\mathsf{POET}_{\pi,F,0,0}(\cdot,\cdot),\mathsf{POET}^{-1}_{\pi^{-1},F,0,0}(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$_{\mathrm{O}}(\cdot,\cdot),\$^{-1}_{\mathrm{O}}(\cdot,\cdot,\cdot)} \Rightarrow 1\right] \right|, \quad (7.7)$$

where $\pi \twoheadleftarrow \mathsf{Perm}_n$ denotes an n-bit random permutation that was chosen uniformly at random from the set of all n-bit random permutations, and π^{-1} denotes its inverse. The difference in Equation (7.6) can be upper bounded by the IND-SPRP-advantage of $\mathcal{A}$ to distinguish E from a random permutation π:

$$\mathbf{Adv}_{E,E^{-1}}^{\mathsf{IND\text{-}SPRP}}(\ell + 4q, \mathrm{O}(t)).$$

The additional term $4q$ results from the fact that the tag-generation of POET requires two additional calls to the block cipher for the encryption of the message length and the tag generation, and the header-processing may involve up to two additional calls to E for encoding the intermediate-tag parameters and another one if the length of the header is a multiple of the block length.

Proof Idea for the Remainder. It remains to study the difference in Equation (7.7), which refers to the advantage of $\mathcal{A}$ to distinguish POET from random. We use a similar argumentation as in our OPERMCCA analysis of POE. Two mutually exclusive cases can occur during the OCCA game: (1) collisions between internal values happen, or (2) no such collisions occur. Again, we define the event COLL to represent the former and the event $\neg$ COLL for the latter case. We define COLLWIN for the event that the adversary wins if at least one collision occurred, and the event NOCOLLWIN for the event that it wins when no collision occurred, respectively. Then, we can derive a simplified upper bound for Equation 7.7 as follows:

$$\Pr[\mathsf{COLL}] \cdot \Pr[\mathsf{COLLWIN}] + \Pr[\neg\,\mathsf{COLL}] \cdot \Pr[\mathsf{NOCOLLWIN}] \leq \Pr[\mathsf{COLL}] + \Pr[\mathsf{NOCOLLWIN}].$$

COLL. In this case, $\mathcal{A}$ tries to distinguish POET from random by exploiting some collision between internal values. Due to the structure of POET, a collision in a chaining value requires that one of the following events must have occurred:

1. $\mathsf{COLL}^{\mathrm{ad}}$: $\mathcal{A}$ found a collision for two distinct headers $H \neq H'$: $\mathbf{ProcessHeader}(H) = \mathbf{ProcessHeader}(H')$.
2. $\mathsf{COLL}^{\mathrm{enc}}$: For two distinct tuples (X_{i-1}, M_i) and (X'_{j-1}, M'_j) in one or two encryption query results $(M, C), (M', C') \in \mathcal{Q}$, the resulting top-row chaining values $X_i = X'_j$ collide.
3. $\mathsf{COLL}^{\mathrm{dec}}$: For two distinct tuples (Y_{i-1}, C_i) and (Y'_{j-1}, C'_j) from one or two decryption query results $(M, C), (M', C') \in \mathcal{Q}$, the resulting bottom-row chaining values $Y_i = Y'_j$ collide.
4. $\mathsf{COLL}^{\mathrm{lmb}}$: For two distinct tuples (X_{i-1}, M_i) and (X'_{j-1}, M'_j) in one or two encryption query results (M, C), $(M', C') \in \mathcal{Q}$, the resulting top-row chaining values $X_i = X'_j$ collide, when M_i or M'_j (or both) is the last block of M and M', respectively.
5. $\mathsf{COLL}^{\mathrm{lcb}}$: For two distinct tuples (Y_{i-1}, C_i) and (Y'_{j-1}, C'_j) in one or two decryption query results (M, C), $(M', C') \in \mathcal{Q}$, the resulting bottom-row chaining values $Y_i = Y'_j$ collide, when C_i or C'_j (or both) is the last block of C and C', respectively.

Moreover, we define COLL as the compound event that represents that any non-empty subset of the events $\mathsf{COLL}^{\mathrm{ad}}$, $\mathsf{COLL}^{\mathrm{enc}}$, $\mathsf{COLL}^{\mathrm{dec}}$, $\mathsf{COLL}^{\mathrm{lmb}}$, $\mathsf{COLL}^{\mathrm{lcb}}$ occurred:

$$\mathsf{COLL} = \mathsf{COLL}^{\mathrm{ad}} \vee \mathsf{COLL}^{\mathrm{enc}} \vee \mathsf{COLL}^{\mathrm{dec}} \vee \mathsf{COLL}^{\mathrm{lmb}} \vee \mathsf{COLL}^{\mathrm{lcb}}. \tag{7.8}$$

We can upper bound the probabilities for these events as follows:

- $\Pr[\mathsf{COLL}^{\mathrm{ad}}] = \frac{5.5(\ell+2q)^2}{2^n}$, which follows from Corollary B.3.
- $\Pr[\mathsf{COLL}^{\mathrm{enc}}] = \frac{(\ell+q)^2}{2}\epsilon$. The proof follows from Corollary B.1.
- $\Pr[\mathsf{COLL}^{\mathrm{lmb}}] = \max\left\{q(\ell+q)\epsilon, \frac{q(\ell+q)}{2^n-q}\right\}$. The proof follows from Lemma B.4.

From the symmetric structure of POET follows that

- $\Pr[\mathsf{COLL}^{\mathrm{dec}}] = \Pr[\mathsf{COLL}^{\mathrm{enc}}]$ and
- $\Pr[\mathsf{COLL}^{\mathrm{lcb}}] = \Pr[\mathsf{COLL}^{\mathrm{lmb}}]$.

NOCOLLWIN. It remains to bound $\Pr[\mathsf{NOCOLLWIN}] = \frac{(\ell+4q)^2}{2^n-(\ell+4q)}$. The proof follows from Lemma A.2. Note that the analysis of the event NOCOLLWIN here is similar to that of the NOCOLLWIN event in Chapter 7.1. It differs only in the fact that for POET one considers $\ell + 4q$ blocks.

Our claim in Theorem 7.2 follows from summing up the individual terms. $\square$

7.3. INT-RUP Security Analysis of POET Without Intermediate Tags

Theorem 7.3 (INT-RUP Security of POET When $\ell_s = 0$). *Let $E \in \mathtt{Block}(k, n)$ be a block cipher and $F : \{0,1\}^n \to \{0,1\}^n$ be an ϵ-AXU family of hash functions. Furthermore, let $F_i : \{0,1\}^{ni} \to \{0,1\}^n$ be an $i\epsilon$-AXU family of hash functions defined as follows:*

$$F_0 = F(1); \quad F_i(M) = F(F_{i-1}(M_1, \ldots, M_{i-1}) \oplus M_i), \qquad i \in \mathbb{N}^+.$$

Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{V})$ be $\mathsf{POET}_{E,F,0,0}$ as defined in Definition 5.1, where the functions $\mathcal{D}$ and $\mathcal{V}$ as defined in Algorithm 7 and it holds that $q \leq \ell \leq 2^{n/2-4}$. Then, it holds that

$$\mathbf{Adv}_{\Pi}^{\textsf{INT-RUP}}(q, \ell, t) \leq \frac{5.5(\ell + 2q)^2}{2^n} + 4q(\ell + 2q)\epsilon + \frac{q}{2^n - (\ell + 2q)} + \frac{q}{2^{n/2+1} - (\ell + 2q)} + \mathbf{Adv}_{E,E^{-1}}^{\textsf{IND-SPRP}}(\ell + 4q, O(t)).$$

Proof. We define $\mathcal{A}$ as an INT-RUP adversary which interacts with three oracles as in Game $G_{\textsf{INT-RUP}}$ (cf. Figure 6.1). $\mathcal{A}$ can ask encryption, decryption, and verification queries to $\mathcal{O}$. We assume that $\mathcal{A}$ stores each of its queries to $\mathcal{O}$ together with the corresponding response as a tuple $Q_i = (H, C, M, T)$ in a query history $\mathcal{Q}$. Since we focus on $\ell_s = \ell_t = 0$, we can omit both parameters in this proof for brevity. For the same reason, the separated functions for decryption and verification in Algorithm 7 omit the decoding since it would not alter the message and since there are no intermediate tags that had to be verified.

$\mathcal{A}$ wins if it can predict the correct authentication tag T for some query (H, C, T) that it has not asked before, i.e., $(H, C, *, T) \notin \mathcal{Q}$ and $\mathcal{V}_K(H, C, T) \neq \perp$. Wlog., we assume that $\mathcal{A}$ does not ask queries to which it already knows the answer. We call a query Q_w that allows $\mathcal{A}$ to set *win* to **true** (cf. Line 25 of Figure 6.1) a *winning query*.

In the remainder, we analyze three possible scenarios that can occur for a winning query of $\mathcal{A}$, depending on

1. whether the header H is fresh, i.e., had been part of a previous query of $\mathcal{A}$.
2. whether the length of the ciphertext $|C|$ of the winning query is a multiple of the block length or not (tag splitting), and
3. whether (C, T) from the winning query had been part of a previous query of $\mathcal{A}$.

The latter two scenarios consider three cases each with regards to the freshness of C and T, as summarized in Table 7.1. It is easy to see that these cases cover all possibilities for $\mathcal{A}$ to win the INT-RUP game.

Scenarios	(1)	(2)			(3)		
		Case (1)	Case (2)	Case (3)	Case (1)	Case (2)	Case (3)
$(H, *, *, *) \in \mathcal{Q}$	○	●	●	●	●	●	●
$(H, C, *, *) \in \mathcal{Q}$	○	○	●	●	○	●	●
$(H, C, *, T) \in \mathcal{Q}$	○	○	○	●	○	○	●

Table 7.1.: Cases for our INT-RUP-security analysis of POET. H, C, and T represent header, ciphertext, and tag of the winning query of $\mathcal{A}$. ● = yes; ○ = no.

For all scenarios, we replace the block cipher E by a random permutation π. Then the advantage for the adversary from this replacement can be upper bounded by

$$\mathbf{Adv}^{\mathsf{IND\text{-}SPRP}}_{E,E^{-1}}(\ell + 4q, \mathrm{O}(t)).$$

Scenario (1): $(H, *, *, *) \notin \mathcal{Q}$. When H is fresh, τ will also be a random output from the **ProcessHeader** step. We distinguish between three cases:

1. τ is old, i.e., $\mathcal{A}$ found a collision $\mathbf{ProcessHeader}(H) = \mathbf{ProcessHeader}(H')$ for two headers $H \neq H'$.
2. τ is fresh and the top-row chaining value computing the tag block, X_{m+1}, occurred before as top-row chaining value X'_j in any query of $\mathcal{A}$.
3. τ is fresh and X_{m+1} of $\mathcal{A}$'s winning query is also fresh.

Case (1): τ Is Old. We let the adversary win if it manages to find a collision of τ for two distinct headers H and H'. As shown in Corollary B.3, the success probability for this event can be upper bounded by

$$\frac{5.5(\ell + 2q)^2}{2^n}.$$

For the following Cases (2) and (3), we assume that no collision of τ for disjoint headers H, H' occurred, and τ is a fresh random value. To forge a tag, the adversary must be able to predict

$$\begin{aligned} T &= Y_{m+1} \oplus F_{K_F}(Y_m) \oplus \tau, \text{ where} \\ Y_{m+1} &= \pi(X_{m+1}), \text{ and} \\ X_{m+1} &= F_{K_F}(X_m) \oplus \tau. \end{aligned}$$

Case (2): τ Is Fresh and X_{m+1} Is Old. In this case, we let $\mathcal{A}$ win since the top-row chaining value X_{m+1} of its winning query collides with some top-row chaining value X'_j of the winning or a previous query of $\mathcal{A}$. τ is a fresh random value and $F_{K_F}(\cdot) \oplus \tau$ is an ϵ-AU family of hash functions. So, this event happens only with probability ϵ for a fixed pair X_{m+1} and X'_j. From the q queries of $\mathcal{A}$, there are q values X_{m+1} that may collide with any of $\ell + 2q$ values X' (including initial values X'_0 and final values $X'_{m'+1}$). So, the probability for this event over all options can be upper bounded by

$$q(\ell + 2q)\epsilon.$$

Case (3): τ Is Fresh and X_{m+1} Is Fresh. In the opposite case, X_{m+1} of $\mathcal{A}$'s winning query is fresh, and thus Y_{m+1} will be a random n-bit value chosen uniformly at random by π from a space of at least $2^n - (\ell + 2q)$ elements for the q-th query, considering ℓ message blocks over all queries plus the q initial chaining values X_0 and the values X_{m+1} for the tag-generation step. Here, we make the adversary stronger than it is by giving it full control over T, $F_{K_F}(Y_m)$, and τ. Then, its chances for correctly guessing Y_{m+1} is, over q queries, at most

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

The total winning probability for $\mathcal{A}$ in Scenario (1) is upper bounded by

$$\frac{5.5(\ell + q)^2}{2^n} + q(\ell + 2q)\epsilon + \frac{q}{2^n - (\ell + 2q)}.$$

Since we already considered the case of a fresh header, we assume for the remaining scenarios that H was queried by $\mathcal{A}$ before.

Scenario (2): No Tag Splitting ($|C_m| = n$). This scenario considers the INT-RUP-security of POET when H is old and the message length is a multiple of n.

Case (1): $(H, C, *, *) \notin \mathcal{Q}$. In this case, we distinguish between two subcases depending on whether the chaining value X_{m+1} of $\mathcal{A}$'s winning query collides with any top-row chaining value X'_j of the same or a previous query of $\mathcal{A}$ or not. Since C is fresh and POET is an on-line permutation, we can upper bound the probability for such a collision by

$$q(\ell + 2q)\epsilon.$$

In the opposite case, X_{m+1} is fresh and therefore Y_{m+1} is a fresh random value that is chosen uniformly at random from a set of size at least $2^n - (\ell + 2q)$. The chance for $\mathcal{A}$ to choose T correctly is at most the probability that $\mathcal{A}$ can predict Y_{m+1}, which is at most

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

Case (2): $(H, C, *, *) \in \mathcal{Q}$, but $(H, C, *, T) \notin \mathcal{Q}$. In this case, we distinguish between two options: Firstly, $\mathcal{A}$ can have queried (H, M) for some message M and obtained (H, C, T') with $T \neq T'$ as the result from the encryption oracle. Then, $\mathcal{A}$'s chances to win with a verification query (H, C, T) are zero since POET maps each tuple (H, C) uniquely to a single tag T:

$$\begin{aligned} T &= \pi(F_{K_F}(X_m) \oplus \tau) \oplus F_{K_F}(Y_m) \oplus \tau, \\ &= Y_{m+1} \oplus F_{K_F}(Y_m) \oplus \tau. \end{aligned}$$

Secondly, we must consider the case that $\mathcal{A}$ did not obtain $(H, C, *)$ from the encryption oracle, but it is contained in the query history since $\mathcal{A}$ asked (H, C, T') as a decryption or verification query before. This reflects that $\mathcal{A}$ tries multiple attempts (H, C, T_1), (H, C, T_2), etc. to guess the correct tag. Since H, C, and therefore also $F_{K_F}(X_m)$, $F_{K_F}(Y_m)$, and τ are fixed and secret, there is only a single value T which is valid. With each rejected query, $\mathcal{A}$ learns an invalid option. Therefore, the chances for $\mathcal{A}$ to correctly predict the tag are at most

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

Case (3): $(H, C, *, T) \in \mathcal{Q}$. Clearly, the advantage of $\mathcal{A}$ in this case is zero since $\mathcal{A}$ either has already asked the tuple (H, C, T) to $\mathcal{O}$ – and was rejected; otherwise, we would have given the attack to $\mathcal{A}$ before – or $\mathcal{A}$ obtained it as response of an encryption query.

Since the cases in Scenario (2) are mutually exclusive, the success probability of $\mathcal{A}$ in this scenario can be upper bounded by

$$\max\left\{ q(\ell + 2q)\epsilon + \frac{q}{2^n - (\ell + 2q)}, \frac{q}{2^n - (\ell + 2q)} \right\} = q(\ell + 2q)\epsilon + \frac{q}{2^n - (\ell + 2q)}.$$

Scenario (3): Tag Splitting. This scenario concerns the INT-RUP-security of POET for messages whose lengths are not multiples of n.

Case (1): $(H, C, *, *) \notin \mathcal{Q}$. First, we upper bound the probability that X_m or Y_m collide with some top- or bottom-row chaining value X'_j or Y'_j of the same or some previous query of $\mathcal{A}$, respectively. Then, $\mathcal{A}$ would have found an internal collision. Since we consider q values X_m and $\ell + 2q$ chaining values X'_j, for $0 \leq j \leq m + 1$, it yields $q(\ell + 2q)$ pairs which can collide each with probability at most ϵ. So, the probability of this event can be upper bounded by

$$2q(\ell + 2q)\epsilon,$$

The same number of possibly colliding pairs are given for the bottom-row chaining values. We simply let $\mathcal{A}$ win if it finds such a collision. For the following, we assume that all chaining values X_m and Y_m are fresh.

Similarly, we also let $\mathcal{A}$ win if it manages that X_{m+1} collides with a previous top-row chaining value X'_j of some queries of $\mathcal{A}$. Since we assume that X_m is fresh and $F_{K_F}(\cdot) \oplus \tau$ is an ϵ-AU family of hash functions, the probability for a single query is upper bounded by

$$\Pr\left[F_{K_F}(X_m) \oplus \tau = X'_j\right] \leq \epsilon,$$

and $q(\ell + 2q)\epsilon$ over q queries with $\ell + 2q$ blocks in total.

Line 807 of procedure **VerifyTag** (see Section 5) defines two conditions which have to be fulfilled both for *win* to be set to **true**:

- $\tau^\alpha = \tau'$ and
- $T^\beta = T'$.

For the sake of readability, we define $\alpha = n - |M_m|$ and $\beta = |M_m|$. First, we consider the probability that the first condition holds. Therefore, $\mathcal{A}$ has to correctly match the α least significant bits of $(M_m \mid\mid \tau^\alpha)$, i.e.,

$$\mathbf{LSB}_\alpha(F_{K_F}(X_{m-1}) \oplus X_m \oplus S) = \tau^\alpha,$$

where $X_m = \pi^{-1}(Y_m)$. Since we assume that Y_m is fresh, its decryption X_m will also be a fresh random value chosen from a set of size at least $2^n - (\ell + 2q)$. Note that for each value $|C|$ the value $S = \pi(|C|)$ is a uniquely defined constant. So, for a fixed query, the probability of this event can be upper bounded by

$$\max_{M_m} \{\Pr\left[F_{K_F}(X_{m-1}) \oplus X_m \oplus S = (M_m \mid\mid \tau^\alpha)\right]\} \leq \frac{1}{2^\alpha - (\ell + 2q)}.$$

For q queries, the probability that $\tau^\alpha = \tau'$ holds is then at most

$$\Pr_\alpha \leq \frac{q}{2^\alpha - (\ell + 2q)}.$$

We can use similar arguments to upper bound the probability of $T^\beta = T'$. For this case, $\mathcal{A}$ must match the β most significant bits of $T^\beta \mid\mid Z$, i.e.,

$$\mathbf{MSB}_\beta(F_{K_F}(Y_m) \oplus Y_{m+1} \oplus \tau) = T^\beta,$$

where $Y_{m+1} = \pi(X_{m+1})$ and $X_{m+1} = F_{K_F}(X_m) \oplus \tau$.

In the following, we assume that X_{m+1} is fresh and so, Y_{m+1} is a fresh random value chosen from a set of size at least $2^n - (\ell + 2q)$. So, the probability for this event for a single query can be upper bounded by

$$\max_{Z} \left\{\Pr\left[F_{K_F}(Y_m) \oplus Y_{m+1} \oplus \tau = (T^\beta \mid\mid Z)\right]\right\} \leq \frac{1}{2^\beta - (\ell + 2q)},$$

and for q queries, the probability that $T^\beta = T'$ is then at most

$$\Pr_\beta \leq \frac{q}{2^\beta - (\ell + 2q)}.$$

Note that the success probability of this case depends on the length of $|C_m|$. We distinguish between the following three subcases:

Subcase (1.1): $|C_m| < n/2$. In this case, we can upper bound $\Pr_\alpha$ by $\frac{1}{2^{n/2+1} - (\ell+2q)}$ and $\Pr_\beta$ by 1. Hence, the total success probability for q queries is at most

$$\Pr_\alpha \cdot \Pr_\beta \leq \frac{q}{2^{n/2+1} - (\ell + 2q)}.$$

Subcase (1.2): $|C_m| = n/2$**.** In this case, we can upper bound $\Pr_\alpha$ by $\frac{1}{2^{n/2}-(\ell+2q)}$ and $\Pr_\beta$ by $\frac{1}{2^{n/2}-(\ell+2q)}$. Hence, the total success probability for q queries is at most

$$\Pr_\alpha \cdot \Pr_\beta \leq \frac{q}{2^{n/2}-(\ell+2q)} \cdot \frac{q}{2^{n/2}-(\ell+2q)} \leq \frac{q^2}{(2^{n/2}-(\ell+2q))^2}.$$

Subcase (1.3): $|C_m| > n/2$**.** In this case, we can upper bound $\Pr_\alpha$ by 1 and $\Pr_\beta$ by $\frac{1}{2^{n/2+1}-(\ell+2q)}$. Hence, the total success probability for q queries is at most

$$\Pr_\alpha \cdot \Pr_\beta \leq \frac{q}{2^{n/2+1}-(\ell+2q)}.$$

Since all three subcases are mutually exclusive, we can upper bound the success probability for $q \leq \ell \leq 2^{n/2-4}$ queries by

$$\max\left\{\frac{q}{2^{n/2+1}-(\ell+2q)}, \frac{q^2}{(2^{n/2}-(\ell+2q))^2}, \frac{q}{2^{n/2+1}-(\ell+2q)}\right\} \leq \frac{q}{2^{n/2+1}-(\ell+2q)}.$$

Then, the total success probability of Case (1) can be upper bounded by

$$2q(\ell+2q)\epsilon + q(\ell+2q)\epsilon + \frac{q}{2^{n/2+1}-(\ell+2q)} = 3q(\ell+2q)\epsilon + \frac{q}{2^{n/2+1}-(\ell+2q)}.$$

Case (2): $(H, C, *, *) \in \mathcal{Q}$ **but** $(H, C, *, T) \notin \mathcal{Q}$**.** In this case, we distinguish between the subcases whether $(H, C, *, T^\alpha \| *) \in \mathcal{Q}$ or $(H, C, *, T^\alpha \| *) \notin \mathcal{Q}$.

If $(H, C, *, T^\alpha \| *) \in \mathcal{Q}$, then X_m and Y_m are old. This can happen because $\mathcal{A}$ has obtained $(H, C, T^\alpha \| *)$ as (part of) the result of an encryption query and modified the corresponding value T^β. Though, POET defines a unique mapping of (H, C, T^α) to T^β:

$$\begin{aligned} Y_m &= F_{K_F}(Y_{m-1}) \oplus (C_m \| T^\alpha) \oplus S \\ X_m &= \pi^{-1}(Y_m) \\ (T^\beta \| Z) &= \pi(F_{K_F}(X_m) \oplus \tau) \oplus (F_{K_F}(Y_m) \oplus \tau). \end{aligned}$$

Thus, the success probability of $\mathcal{A}$ is zero. The subcase that $\mathcal{A}$ did not obtain (H, C, T^α) from an encryption query is handled in a moment.

Secondly, we investigate the subcase $(H, C, *, T^\alpha \| *) \notin \mathcal{Q}$. We define that $\mathcal{A}$ wins if it manages to find a collision of Y_m with some previous bottom-row chaining value Y'_j of the same or a previous query of $\mathcal{A}$. This probability can be upper bounded by

$$q(\ell+2q)\epsilon.$$

If Y_m is fresh, it follows that $X_m = \pi^{-1}(Y_m)$ is also a fresh random value chosen at random from a set of size at most $2^n - (\ell+2q)$ elements. In order that $\mathcal{A}$ wins the INT-RUP game, the same two conditions $\tau^\alpha = \tau'$ and $T^\beta = T'$ as in Case (1) must be fulfilled; these hold with the same success probabiliy as in Case (1):

$$\frac{q}{2^{n/2+1}-(\ell+2q)}.$$

Finally, if $\mathcal{A}$ has not obtain $(H, C, T^\alpha \| *)$ as the result of a previous encryption query, $\mathcal{A}$ actually tries to guess T. Again, the same conditions as in Case (1) must hold for τ^α and T^β, with a success probability of at most

$$\frac{q}{2^{n/2+1}-(\ell+2q)}.$$

The success probability of Case (2) can then be upper bounded by

$$q(\ell+2q)\epsilon + \frac{q}{2^{n/2+1}-(\ell+2q)}.$$

Case (3): $(H, C, *, T) \in \mathcal{Q}$**.** Clearly, the advantage of $\mathcal{A}$ is zero in this case.

Since the cases of Scenario (3) are mutually exclusive, we can upper bound the advantage for $\mathcal{A}$ in Scenario (3) by

$$\begin{aligned} &\max\left\{3q(\ell+2q)\epsilon + \frac{q}{2^{n/2+1} - (\ell+2q)}, q(\ell+2q)\epsilon + \frac{q}{2^{n/2+1} - (\ell+2q)}\right\} \\ =\ &3q(\ell+2q)\epsilon + \frac{q}{2^{n/2+1} - (\ell+2q)}. \end{aligned}$$

Finally, our claim in Theorem 7.3 follows then from the sum of individual terms and the maximum of the probabilities of the mutually exclusives Scenarios (2) and (3). Since the probability in Scenario (2) is equal or lower than the probability in Scenario (3), we have to write only the latter in the final bound. □

7.4. OCPA-IT Security Analysis of POET With Intermediate Tags

Theorem 7.4 (OCPA-IT Security of POET When $\ell_s \geq 1$ and $\ell_t = n$). *Let $E \in \texttt{Block}(k,n)$ be a block cipher and $F : \{0,1\}^n \to \{0,1\}^n$ an ϵ-AXU family of hash functions. Furthermore, let $F_i : \{0,1\}^{ni} \to \{0,1\}^n$ be an $i\epsilon$-AXU family of hash functions defined as follows:*

$$F_0 = F(1); \quad F_i(M) = F(F_{i-1}(M_1, \ldots, M_{i-1}) \oplus M_i), \qquad i \in \mathbb{N}^+.$$

Let $\Pi = (\mathcal{K}, \widetilde{\mathcal{E}}, \widetilde{\mathcal{D}})$ be $\mathsf{POET}_{E,F,\ell_s,\ell_t}$ as defined in Definition 5.1, with $\ell_s \geq 1$ and $\ell_t = n$. Then, it holds that

$$\mathbf{Adv}_{\Pi}^{\mathsf{OCPA\text{-}IT}}(q,\ell,t) \leq \frac{5.5(\ell+2q)^2}{2^n} + (2\ell+q)^2\epsilon + 2\max\left\{q(2\ell+q)\epsilon, \frac{q(2\ell+q)}{2^n-q}\right\}$$
$$+ \frac{(2\ell+4q)^2}{2^n-(2\ell+4q)} + \mathbf{Adv}_{E,E^{-1}}^{\mathsf{IND\text{-}SPRP}}(2\ell+4q, O(t)).$$

Proof Idea. The proof follows the ideas of our previous proofs of Theorems 7.1 and 7.2.

We consider the encryption and decryption algorithms of POET from Algorithm 2 in Section 5.1. We define $\mathcal{A}$ as an OCPA-IT adversary that asks at most q queries of at most ℓ blocks in total, where the intermediate tags are not counted as part of to the ℓ blocks. Throughout the proof, we will denote by σ the number of blocks for intermediate tags summed over all queries by $\mathcal{A}$. $\mathcal{A}$ stores each query along with the corresponding response as a tuple $(\ell_s, \ell_t, H, M, C, T)$ in a query history $\mathcal{Q}$. Wlog. $\mathcal{A}$ will not make queries to which it already knows the answer. Here, we focus on $\ell_s \geq 1$ and $\ell_t = n$ from our recommendation for POET.

Again, we first replace the block cipher E by an n-bit random permutation. The difference between both settings can be upper bounded by the maximal IND-SPRP-advantage of an adversary which runs in time at most $O(t)$ and asks at most $\ell + \sigma + 4q$ queries for distinguishing E, E^{-1} from a random permutation,

$$\mathbf{Adv}_{E,E^{-1}}^{\mathsf{IND\text{-}SPRP}}(\ell + \sigma + 4q, \mathrm{O}(t)).$$

As before, two mutually exclusive cases can occur during the game; we denote by COLL the event that some internal collision occured during the game, and by $\neg$ COLL the opposite event. COLLWIN and NOCOLLWIN represent again the events that the adversary wins by finding at least one internal collision or that it wins when no internal collision occurred, respectively. We are interested in upper bounding the success probability of $\mathcal{A}$ by

$$\Pr[\mathsf{COLL}] \cdot \Pr[\mathsf{COLLWIN}] + \Pr[\neg\,\mathsf{COLL}] \cdot \Pr[\mathsf{NOCOLLWIN}] \leq \Pr[\mathsf{COLL}] + \Pr[\mathsf{NOCOLLWIN}].$$

Assume that $\mathcal{A}$ has already found an internal collision for two of its queries $Q = (\ell_s, \ell_t, H, M, C, T)$, $Q' = (\ell'_s, \ell'_t, H', M', C', T')$. Then we define by EQ_SLEN the event that the part lengths of these queries are equal, i.e., $\ell_s = \ell'_s$, and by $\neg$ EQ_SLEN the opposite event, i.e., $\ell_s \neq \ell'_s$. Since both cases are mutually exclusive, it follows that

$$\Pr[\mathsf{COLL}] + \Pr[\mathsf{NOCOLLWIN}] \leq \max\{\Pr[\mathsf{COLL} \mid \mathsf{EQ\_SLEN}], \Pr[\mathsf{COLL} \mid \neg\,\mathsf{EQ\_SLEN}]\}$$
$$+ \Pr[\mathsf{NOCOLLWIN}].$$

In the following, we study three cases for which we bound the probability: First, we concern the probability that $\mathcal{A}$ finds an internal collision for two of its queries Q and Q' with equal part lengths, i.e., $\ell_s = \ell'_s$; thereupon, we bound the probability that $\mathcal{A}$ finds an internal collision for two queries with different part lengths $\ell_s \neq \ell'_s$; finally, we bound the probability for the event NOCOLLWIN.

Case (1): Internal Collision for Two Queries Q and Q' with Equal part Lengths. The same encoding and decoding functions are used in both the real and random setting. Here, we can use that the encoding is key-independent and injective. The key independence allows that for any given ℓ_s, ℓ_t, header, and message, the adversary can always compute the encoded header and message without interaction with any oracle. The injectivity ensures that the encoding is reversible, i.e., there are no two tuples (ℓ_s, ℓ_t, H, M) and $(\ell'_s, \ell'_t, H', M')$ that map to the same encoded header and message. It follows directly, that the advantage of an $\mathsf{OCPA\text{-}IT}$ adversary $\mathcal{A}$ on POET with intermediate tags is at most that of an OCPA adversary $\mathcal{B}$ on POET without intermediate tags, where $\mathcal{B}$ has full control over the encoding of the parameters into the header and runs in time proportional to $O(t)$ and asks the same number of q queries of $\mathcal{A}$ with $\ell + \sigma$ blocks.

So, the same events for internal collisions as in our OCCA on POET can be applied here, and we can borrow the events $\mathsf{COLL}^{\mathrm{ad}}$, $\mathsf{COLL}^{\mathrm{enc}}$, $\mathsf{COLL}^{\mathrm{dec}}$, $\mathsf{COLL}^{\mathrm{lmb}}$, or $\mathsf{COLL}^{\mathrm{lcb}}$, which represent all possibilities for internal collisions:

$$\begin{aligned}\Pr[\mathsf{COLL} \mid \mathsf{EQ\_SLEN}] \leq\ & \Pr[\mathsf{COLL}^{\mathrm{ad}} \mid \mathsf{EQ\_SLEN}] + \Pr[\mathsf{COLL}^{\mathrm{enc}} \mid \mathsf{EQ\_SLEN}] \\ & + \Pr[\mathsf{COLL}^{\mathrm{dec}} \mid \mathsf{EQ\_SLEN}] + \Pr[\mathsf{COLL}^{\mathrm{lmb}} \mid \mathsf{EQ\_SLEN}] \\ & + \Pr[\mathsf{COLL}^{\mathrm{lcb}} \mid \mathsf{EQ\_SLEN}].\end{aligned}$$

Though, we must take the number of intermediate blocks into account, which yields the following probabilities.

- $\Pr[\mathsf{COLL}^{\mathrm{ad}} \mid \mathsf{EQ\_SLEN}] = \frac{5.5(\ell+2q)^2}{2^n}$. The proof follows from Corollary B.3.
- $\Pr[\mathsf{COLL}^{\mathrm{enc}} \mid \mathsf{EQ\_SLEN}] = \frac{(\ell+\sigma+q)^2}{2} \cdot \epsilon$. The proof follows from Corollary B.1 and the fact that we consider σ additional intermediate-tag blocks.
- $\Pr[\mathsf{COLL}^{\mathrm{lmb}} \mid \mathsf{EQ\_SLEN}] = \max\left\{(\ell+\sigma+q)q\epsilon, \frac{(\ell+\sigma+q)q}{2^n-q}\right\}$. The proof follows from Lemma B.4 and the fact that we consider σ additional intermediate-tag blocks.
- $\Pr[\mathsf{COLL}^{\mathrm{dec}} \mid \mathsf{EQ\_SLEN}] = \Pr[\mathsf{COLL}^{\mathrm{enc}} \mid \mathsf{EQ\_SLEN}]$ follows from the symmetric structure of POET.
- $\Pr[\mathsf{COLL}^{\mathrm{lcb}} \mid \mathsf{EQ\_SLEN}] = \Pr[\mathsf{COLL}^{\mathrm{lmb}} \mid \mathsf{EQ\_SLEN}]$ follows from the symmetric structure of POET.

Case (2): Internal Collision for Two Queries Q and Q' with Different part Lengths. Here, we can follow a similar argumentation as in the first case. The different part lengths ℓ_s and ℓ'_s imply always different headers $\widetilde{H} = (\langle \ell_s \rangle_x \| \langle \ell_t \rangle_x \| H)$ and $\widetilde{H}' = (\langle \ell'_s \rangle_x \| \langle \ell_t \rangle_x \| H')$. Thus, the queries $(\widetilde{H}, \widetilde{M})$ and $(\widetilde{H}', \widetilde{M}')$ never share a common prefix after encoding. In the random world, the choice of the permutation depends on $\widetilde{H}$ and therefore is chosen independently at random for Q and Q'. As a result, the advantage for $\mathcal{A}$ to distinguish between two worlds is upper bounded by the OCCA advantage of $\mathcal{A}$ on POET by finding an internal collision for two queries which have no common prefix. Therefore, the same events as in the OCCA proof on POET apply to bound this probability

$$\begin{aligned}\Pr[\mathsf{COLL} \mid \neg\,\mathsf{EQ\_SLEN}] \leq\ & \Pr[\mathsf{COLL}^{\mathrm{ad}} \mid \neg\,\mathsf{EQ\_SLEN}] + \Pr[\mathsf{COLL}^{\mathrm{enc}} \mid \neg\,\mathsf{EQ\_SLEN}] \\ & + \Pr[\mathsf{COLL}^{\mathrm{dec}} \mid \neg\,\mathsf{EQ\_SLEN}] + \Pr[\mathsf{COLL}^{\mathrm{lmb}} \mid \neg\,\mathsf{EQ\_SLEN}] \\ & + \Pr[\mathsf{COLL}^{\mathrm{lcb}} \mid \neg\,\mathsf{EQ\_SLEN}].\end{aligned}$$

Though, the probabilities for the individual events must take the number of intermediate blocks into account, which give the probabilities from Case (1).

Case (3): $\mathsf{NOCOLLWIN}$. It remains to bound $\Pr[\mathsf{NOCOLLWIN}]$. Since the same encoding is used for both the real and the random setting, the success probability of $\mathcal{A}$ for distinguishing POET

from random when no internal collisions occurred, follows from Lemma A.2, where we consider $\ell + \sigma + 4q$ calls to the block cipher over all queries of $\mathcal{A}$, which gives

$$\Pr[\mathsf{NOCOLLWIN}] = \frac{(\ell + \sigma + 4q)^2}{2^n - (\ell + \sigma + 4q)}.$$

Our claim in Theorem 7.5 follows from summing up the individual terms. Since the number of message blocks between any two subsequent intermediate tags is at least $\ell_s \geq 1$, we use the fact that $\sigma \leq \ell$ to simplify the bound by replacing $\ell + \sigma$ by 2ℓ. $\square$

7.5. INT-CTXT-IT Security Analysis of POET with Intermediate Tags

Theorem 7.5 (INT-CTXT-IT Security of POET for $\ell_s \geq 1$ and $\ell_t = n$). *Let $E \in \texttt{Block}(k,n)$ be a block cipher and $F : \{0,1\}^n \to \{0,1\}^n$ be an ϵ-AXU family of hash functions. Furthermore, let $F_i : \{0,1\}^{ni} \to \{0,1\}^n$ be an $i\epsilon$-AXU family of hash functions defined as follows:*

$$F_0 = F(1); \quad F_i(M) = F(F_{i-1}(M_1, \ldots, M_{i-1}) \oplus M_i), \qquad i \in \mathbb{N}^+.$$

Let $\Pi = (\mathcal{K}, \widetilde{\mathcal{E}}, \widetilde{\mathcal{D}})$ be $\mathsf{POET}_{E,F,\ell_s,\ell_t}$ as defined in Definition 5.1 with $\ell_s \geq 1$ and $\ell_t = n$. Then, it holds that

$$\begin{aligned}\mathbf{Adv}_{\Pi}^{\textsf{INT-CTXT-IT}}(q,\ell,t) \leq\ & \frac{5.5(\ell+2q)^2}{2^n} + (2\ell+2q)^2\epsilon + \frac{\sigma}{2^n-(2\ell+2q)} \\ & + \ell(2\ell+2q)\epsilon + \frac{\ell}{2^n-(2\ell+2q)} \\ & + \mathbf{Adv}_{\mathsf{POET}_{E,E^{-1},F,0,0}}^{\textsf{INT-RUP}}(2\ell,q,t) + \mathbf{Adv}_{E,E^{-1}}^{\textsf{IND-SPRP}}(2\ell+4q, O(t)).\end{aligned}$$

Proof Idea. We define $\mathcal{A}$ as an INT-CTXT-IT adversary with access to an oracle as in Game $G_{\textsf{INT-CTXT-IT}}$. $\mathcal{A}$ can ask encryption or decryption queries to its oracle. We assume that $\mathcal{A}$ stores each of its queries together with the corresponding response as a tuple $Q_i = (\ell_s, \ell_t, H, C, M, T)$ in a query history $\mathcal{Q}$. $\mathcal{A}$ wins if it can predict a correct intermediate or authentication tag for some query. Wlog., we assume that $\mathcal{A}$ does not ask queries to which it already knows the answer. We call a query a winning query if it allows $\mathcal{A}$ to set win to **true** in Line 25 in the $G_{\textsf{INT-CTXT-IT}}$ game (see Figure 6.4).

First, we replace E by a random permutation π. The advantage of the adversary to distinguish this replacement can be upper bounded as in our previous proofs by

$$\mathbf{Adv}_{E,E^{-1}}^{\textsf{IND-SPRP}}(\ell+\sigma+4q, \mathrm{O}(t)).$$

The adversary has two options to win: Firstly, it can forge a final tag. The probability for this event can be upper bounded by the maximal INT-CTXT advantage of an adversary on POET without intermediate tags, adding the number of intermediate-tag blocks which are not counted as part of the ℓ blocks that the adversary may ask to its oracles. Since INT-RUP security clearly implies INT-CTXT security, we can upper bound this by

$$\mathbf{Adv}_{\mathsf{POET}_{\pi,\pi^{-1},F,0,0}}^{\textsf{INT-RUP}}(\ell+\sigma, q, t).$$

Secondly, the adversary can win by forging some intermediate tag. For this purpose, we distinguish further between two mutually exclusive scenarios for the winning query of $\mathcal{A}$, depending on whether:

- the encoded header $\widetilde{H}$ is fresh, i.e., has not been part of a previous query of $\mathcal{A}$, or
- the header $\widetilde{H}$ is old.

Clearly, these cover all possible cases. Since the encoding of the parameters into the header is injective, it follows that if H is fresh, then the corresponding encoded header $\widetilde{H}$ is also fresh.

Prior, we define a mapping of the block indices from the unencoded to the encoded message and vice versa. We assume that $\mathcal{A}$ tries to forge the i-th intermediate tag of a ciphertext $C = (C_1, C_2, \ldots)$. From the encoding used within POET, the i-th intermediate tag corresponds to the ciphertext block $C_{i\cdot(\ell_s+1)}$. We will refer to this block also as T_i and introduce an injective index-mapping function $f(i) := i(\ell_s+1)$.

As shown in Line 603 in the procedure **DecodePart** from Algorithm 2, the i-th intermediate tag is valid if and only if it holds for the decryption of the corresponding i-th encoded message block that $\widetilde{M}_{f(i)} = 0^n$. We denote by $X_{f(i)}$ and $Y_{f(i)}$ the top- and bottom-row chaining values at the $f(i)$-th block and by $X_{f(i)-1}$ and $Y_{f(i)-1}$ the chaining values at the preceding block.

Scenario (1): $(\ell_s, \ell_t, H, *, *, *) \notin \mathcal{Q}$. When H is fresh, then τ will be a random output from the **ProcessHeader** step. We distinguish between three cases:

1. τ is old. This means, $\mathcal{A}$ found a collision **ProcessHeader**$(\widetilde{H})$ = **ProcessHeader**$(\widetilde{H}')$ for two distinct encoded headers $\widetilde{H}$ and $\widetilde{H}'$.
2. τ is fresh and the chaining value $X_{f(i)}$ at the position of the intermediate tag in $\mathcal{A}$'s winning query occurred before as bottom-row chaining value Y'_j in the same or a previous query of $\mathcal{A}$.
3. τ is fresh and $Y_{f(i)}$ of $\mathcal{A}$'s winning query is also fresh.

Case (1): τ is Old. We let the adversary win if it manages to find a collision of τ for two distinct headers H and H'. As shown in Corollary B.3, the success probability for this event can be upper bounded by

$$\frac{5.5(\ell + 2q)^2}{2^n}.$$

For the following Cases (2) and (3), we assume that no collision of τ occurred. To forge the i-th intermediate tag, the adversary must be able to predict

$$\begin{aligned} C_{f(i)} = T_i &= Y_{f(i)} \oplus F_{K_F}(Y_{f(i)-1}), \text{ where} \\ Y_{f(i)} &= \pi(X_{f(i)}), \text{ and} \\ X_{f(i)} &= F_{K_F}(X_{f(i)-1}) \oplus \tau. \end{aligned}$$

Case (2): τ is Fresh and $Y_{f(i)}$ is Old. In this case, we let $\mathcal{A}$ win since $Y_{f(i)}$ collides with some bottom-row chaining value of the winning or a previous query of $\mathcal{A}$. This can be because of This event can happen due to two mutually exclusive subcases: $Y_{f(i)-1}$ is old, or $Y_{f(i)-1}$ is fresh. From the fact that τ is fresh follows that $Y_{f(i)-1}$ is not part of a common prefix. So, if $Y_{f(i)-1}$ is old in the first subcase, then $\mathcal{A}$ would have already found a collision of the chaining values for some preceding block in the winning query. In the latter subcase, if $Y_{f(i)-1}$ is fresh, then $\mathcal{A}$ found a collision of the chaining value at the $f(i)$-th block. For both subcases, we can generalize that there exists some block Y_j in $\mathcal{A}$'s winning query which is the first bottom-row chaining value $\mathcal{A}$'s winning query that collides with a $Y'_{j'}$ of some previous query of $\mathcal{A}$. This means that Y_{j-1} and $Y'_{j'-1}$ are different. Since Y_j and $Y'_{j'}$ are computed by

$$\begin{aligned} Y_j &= F_{K_F}(Y_{j-1}) \oplus C_j \text{ and} \\ Y'_{j'} &= F_{K_F}(Y'_{j'-1}) \oplus C'_{j'}, \end{aligned}$$

it must hold for a collision that

$$F_{K_F}(Y_{j-1}) \oplus F_{K_F}(Y'_{j'-1}) = C_j \oplus C'_{j'}.$$

Since $F_{K_F}(\cdot)$ is an ϵ-AXU family of hash functions, it follows from Theorem 4.3 that $F'_{K_F}(x, y) := F_{K_F}(x) \oplus y$ is an ϵ-AU family of hash functions. So, the probability for a collision at fixed blocks is at most ϵ. Over q queries with ℓ message blocks and $2q$ queries for the initial chaining values X_0/Y_0, those for the tag-generation step X_{m+1}/Y_{m+1}, and the σ intermediate-tag blocks, the probability for a collision can be upper bounded by

$$(\ell + \sigma + 2q)^2 \epsilon.$$

Case (3): τ Is Fresh and $Y_{f(i)}$ Is Fresh. In this case, $Y_{f(i)}$ of $\mathcal{A}$'s winning query is fresh and thus, $X_{f(i)}$ will be a random n-bit value chosen uniformly at random by π^{-1} from a space of at least $2^n - (\ell + \sigma + 2q)$ elements. Here, we make the adversary stronger than it is by giving it full control over T_i and $F_{K_F}(Y_{f(i)-1})$. Then, its chances for correctly guessing $Y_{f(i)}$ is, over all queries, at most

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

The total winning probability for $\mathcal{A}$ in Scenario (1) is upper bounded by

$$\frac{5.5(\ell + 2q)^2}{2^n} + (\ell + \sigma + 2q)^2 \epsilon + \frac{\sigma}{2^n - (\ell + \sigma + 2q)}.$$

Scenario (2): $(\ell_s, \ell_t, H, *, *, *) \in \mathcal{Q}$. In this scenario, we consider that the encoded header is old, but the part which contains the intermediate tag that $\mathcal{A}$ forged in its winning query is not part of a common prefix with previous queries of $\mathcal{A}$.

Case (1): $(\ell_s, \ell_t, H, C_1 \mid\mid \ldots \mid\mid C_{f(i)-1} \mid\mid *, *, *) \notin \mathcal{Q}$. We distinguish between two subcases depending on (1) whether the chaining value $Y_{f(i)}$ of $\mathcal{A}$'s winning query collides with any bottom-row chaining value Y'_j of the same or a previous query of $\mathcal{A}$ or (1) $Y_{f(i)}$ is fresh.

In the former subcase, from the fact that $C_{f(i)}$ is fresh follows that

$$\begin{aligned} Y_{f(i)} &= Y'_j \\ C_{f(i)} \oplus F_{K_F}(Y_{f(i)-1}) &= C'_j \oplus F_{K_F}(Y'_{j-1}) \\ F_{K_F}(Y_{f(i)-1}) \oplus F_{K_F}(Y'_{j-1}) &= C_{f(i)} \oplus C'_j. \end{aligned}$$

Since $F'_{K_F}(x, y) := F_{K_F}(x) \oplus y$ is an ϵ-AU family of hash functions, the probability for this event is at most ϵ. Over all queries by $\mathcal{A}$, there are σ intermediate-tag blocks whose chaining values may collide with any of the at most $\ell + \sigma + 2q$ blocks. Hence, we can upper bound the probability for such a collision over all queries by

$$\sigma(\ell + \sigma + 2q)\epsilon.$$

In the opposite case, $Y_{f(i)}$ is fresh. Therefore, $X_{f(i)}$ is a fresh random value that is chosen uniformly at random from a set of size at least $2^n - (\ell - \sigma + 2q)$. The chance for $\mathcal{A}$ to choose T_i correctly is at most the probability that $\mathcal{A}$ can predict $X_{f(i)}$. Over σ intermediate blocks of all queries of $\mathcal{A}$, the success probability of $\mathcal{A}$ can be upper bounded by

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

Case (2): $(\ell_s, \ell_t, H, C_1 \mid\mid \ldots \mid\mid C_{f(i)-1} \mid\mid *, *, *) \in \mathcal{Q}$ but $(\ell_s, \ell_t, H, C_1 \mid\mid \ldots \mid\mid C_{f(i)} \mid\mid *, *, *) \notin \mathcal{Q}$. We distinguish again between two subcases. Firstly, $\mathcal{A}$ can have asked a query, that is encoded to $(\widetilde{H}, \widetilde{M_1} \mid\mid \ldots \mid\mid \widetilde{M}_{f(i)} \mid\mid *)$ and obtained $(C_1 \mid\mid \ldots \mid\mid C'_{f(i)} \mid\mid *, *)$ as ciphertext, with a different intermediate tag than in $\mathcal{A}$'s winning query. $C'_{f(i)} \neq C_{f(i)}$. Then, $\mathcal{A}$'s chances to win with a verification query $(\ell_s, \ell_t, H, C_1 \mid\mid \ldots \mid\mid C_{f(i)} \mid\mid *)$ are zero since POET maps $(\widetilde{H}, \widetilde{M_1} \mid\mid \ldots \mid\mid \widetilde{M}_{f(i)} \mid\mid *)$ to a unique value $C'_{f(i)}$.

Secondly, we must consider the option that $\mathcal{A}$ did not obtain $(\ell_s, \ell_t, H, C_1 \mid\mid \ldots \mid\mid C_{f(i)-1} \mid\mid *, *, *)$ from the encryption oracle, but it was a prefix of a previous decryption query of $\mathcal{A}$. This reflects that $\mathcal{A}$ tries multiple attempts to guess the correct intermediate tag. Since there is only a single value $C_{f(i)}$ which is valid, $\mathcal{A}$ learns an invalid option with every rejected query. Therefore, the chances for $\mathcal{A}$ to correctly predict the tag are at most

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

Case (3): $(\ell_s, \ell_t, H, C_1 \mid\mid \ldots \mid\mid C_{f(i)} \mid\mid *, *, *) \in \mathcal{Q}$. In this case, the advantage of $\mathcal{A}$ is zero. If the prefix was part of a previous decryption query, then $\mathcal{A}$ was either rejected for $C_{f(i)}$ or we would have given the attack to $\mathcal{A}$ before. Clearly, $\mathcal{A}$'s advantage is zero in both cases.

Alternatively, $\mathcal{A}$ may have obtained it (as part of) a response of an encryption query. Then, its chances to win with $C_{f(i)}$ are again zero since it is only part of a common prefix of at least one of $\mathcal{A}$'s previous queries.

Since the cases in Scenario (2) are mutually exclusive, the winning probability for $\mathcal{A}$ in this scenario is upper bounded by

$$\max\left\{\sigma(\ell+\sigma+2q)\epsilon + \frac{\sigma}{2^n-(\ell+\sigma+2q)}, \frac{\sigma}{2^n-(\ell+\sigma+2q)}\right\} = \sigma(\ell+\sigma+2q)\epsilon + \frac{\sigma}{2^n-(\ell+\sigma+2q)}.$$

Our bound in Theorem 7.5 follows from the sum of all terms. For the sake of simplicity, we use the fact that $\sigma \leq \ell$ to replace σ by ℓ.

Chapter 8

Implementation

8.1. Encoding Conventions

All values are encoded as octet strings, and in the following denoted as byte array. A block `b` consists of 16 octets, where `b[0]` denotes the most and `b[15]` the least significant octet. For example, the 128-bit value 448 (i.e., `0x1c0`) will be encoded in a block as $\texttt{b[0]} = \ldots \texttt{b[13]} = \texttt{0}$, $\texttt{b[14]} = \texttt{0x01}$, $\texttt{b[15]} = \texttt{0xc0}$. To refer to individual bits, we denote by $\texttt{b}_\texttt{i}$ the i-th least significant bit of `b`.

However, when a block is used as an element in GF(2^{128}), we borrow the encoding from GCM. This means that for $\texttt{a}_0\texttt{x}^0 + \texttt{a}_1\texttt{x}^1 + \ldots + \texttt{a}_{127}\texttt{x}^{127}$ we have $\texttt{b}_\texttt{i} = \texttt{a}_\texttt{i}$, e.g., $\texttt{x}^6 + \texttt{x}^7 + \texttt{x}^8$ will be encoded as $\texttt{b[0]} = \texttt{0x03}, \texttt{b[1]} = \texttt{0x80}, \texttt{b[2]} = \ldots = \texttt{b[15]} = \texttt{0}$.

The length of the message $|M|$ is represented as 128-bit little-endian-encoded integer. For example, the message length 448 is encoded as $\texttt{b[0]} = \texttt{0xc0}$, $\texttt{b[1]} = \texttt{0x01}$, $\texttt{b[2]} = \ldots = \texttt{b[15]} = \texttt{0}$.

Remark. POE and POET can be implemented efficiently in both software and hardware. Though, our reference implementation of POET is not supposed to be optimized for the large variety of supported platforms. We assume that, in the majority of cases, the block cipher will be AES, and the hash function will be reduced-round AES. Therefore, this section recalls the state-of-the-art for implementations of the AES on 8-, 32-, and 64-bit processors with and without native instructions (NIs). Thereupon, we present figures for optimized versions from our and existing third-party implementations.

8.2. Software Performance of POET

POET implementations using AES-NI. In 2010, Intel [26] introduced native instructions for the AES encryption and decryption, which are nowadays supported by all modern Intel (Sandy Bridge, Ivy Bridge, Haswell, and Broadwell series) and AMD (Bulldozer, Piledriver, and Jaguar series) processors. Intel's *AES New Instruction Set* consists of six constant-time CPU instructions: `aesenc`, `aesenclast`, `aesdec`, `aesdeclast`, `aesimc`, and `aeskeygenassist` for faster key scheduling. On Haswell processors, the `aesenc`, `aesenclast`, `aesdec`, and `aesdeclast` possess a throughput of a single and a latency of 7 cycles [24].

On current x86/x64 processors with AES-NI, the structure of POET allows to compute the top, ECB, and bottom layers in parallel. The overall performance of POET is therefore bounded by the latency of the sequential part, i.e., that of the top layer in encryption and that of the bottom layer in decryption direction.

It is easy to determine a lower bound for the number of cycles from the characteristics of the AES-NI round instructions. Since they possess a latency of seven cycles each, the ten rounds of the top layer in POET-AES10-AES10 require at least 70 cycles plus a small overhead for loading the next input block and XORing it into the state. Using similar arguments, we can calculate that each hash-function call in POET-AES10-AES4 incurs a latency of $4 \cdot 7 = 28$ cycles per block plus the above considerations. We can eliminate the overhead by performing the load and XOR operations parallel to the encryption, which–to the best of our knowledge–has been proposed first by Gregg and Manley [38]. This allows to process an AES block with 70 cycles per block or 4.375 cycles per byte and four-round AES with 28 cycles per block or 1.75 cycles per byte. Our implementations of POET employ this strategy. Table 8.1 and Figure 8.1 show the performance for POET-AES10-AES10 and POET-AES10-AES4 with full setup and for authenticated encryption and decryption[1], respectively. We measured the performance on an Intel i5-4200M (Haswell) with 3 MB L1 cache and running at 2.50 GHz, and with Intel TurboBoost, HyperThreading, and SpeedStep *disabled*. Each measurement represents the median of 10 000 runs, where our C implementation was compiled using `clang 3.5.2-1` with options `-O3 -msse4.2 -mavx2 -maes -march=native`.

For long messages, our implementations achieve 4.39 and 1.77 cycles per byte, respectively, which is fairly close to the optimum. Nevertheless, our measured results for small messages indicate still potential for future implementations to optimize the key setup further.

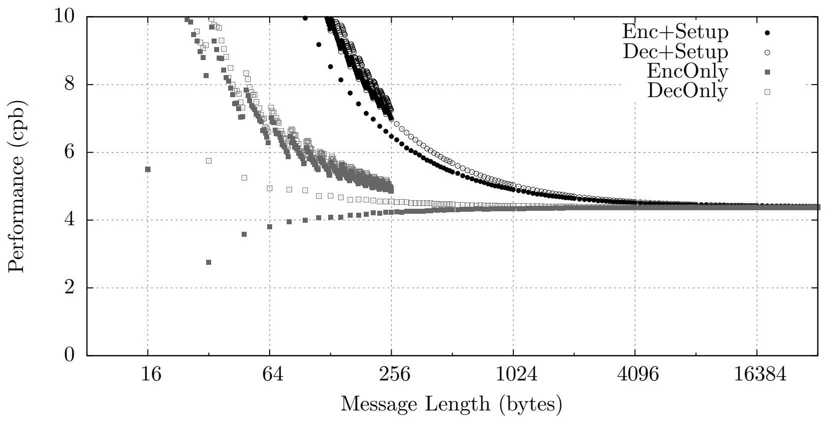
	Message length (bytes)						
Version	128	256	512	1024	2048	4096	32768
POET-AES10-AES10							
Enc+Setup	8.53	6.47	5.42	4.90	4.64	4.51	4.39
Dec+Setup	9.59	6.98	5.68	5.03	4.70	4.54	4.40
Enc Only	5.71	4.93	4.65	4.34	4.36	4.37	4.37
Dec Only	5.77	4.90	4.64	4.42	4.40	4.39	4.38
POET-AES10-AES4							
Enc+Setup	5.88	3.81	2.77	2.27	2.01	1.88	1.77
Dec+Setup	6.44	4.12	2.94	2.35	2.05	1.90	1.77
Enc Only	2.64	2.19	1.98	1.78	1.76	1.76	1.75
Dec Only	2.64	2.19	2.00	1.81	1.78	1.76	1.75

Table 8.1.: Speed measurements in cycles per bytes for optimized implementations of POET without intermediate tags on Haswell.

AES implementations without AES-NI. Using standard optimization techniques (such as combined shift-and-mask instructions, scaled-index loads, etc.), Bernstein and Schwabe showed in 2008 [10] that the AES can be implemented for a 64-bit architecture without native instructions to run at about 10 clock cycles per byte (cpb). In 2009, Käsper and Schwabe [21] improved these results with a bitsliced implementation of the AES that exploited the *Streaming SIMD Extension instructions* (`SSE1-SSE4.2`). Their implementations achieved about 7.59 cpb on an Intel Core2 Q6600 (Kentsfield) and about 6.92 cpb on a Intel i7-920 CPU (Bloomfield). For 32-bit processors, Bernstein and Schwabe achieved a throughput of about 14.13 cpb on a Pentium 4 f12 (Willamette). For an example of a mobile 32-bit processor, we can use the figures for the ARM Cortex-A8 by Krovetz and Rogaway [37].

Concerning off-the-shelf 8-bit processors, Osvik et al. [40] showed at FSE 2010 that the AES can be implemented at speeds of around 124.6 cpb on an AVR Atmel. Considering slightly modified devices, Tillich and Herbst [55] proposed an enhancement for 8-bit AVR cores, which allow to perform an AES encryption at a speed of about 78.7 cpb (1,259 cycles/block) on an Atmel AT-

[1]This means, including the costs for generating the authentication tag, but omitting the costs for key setup and header processing



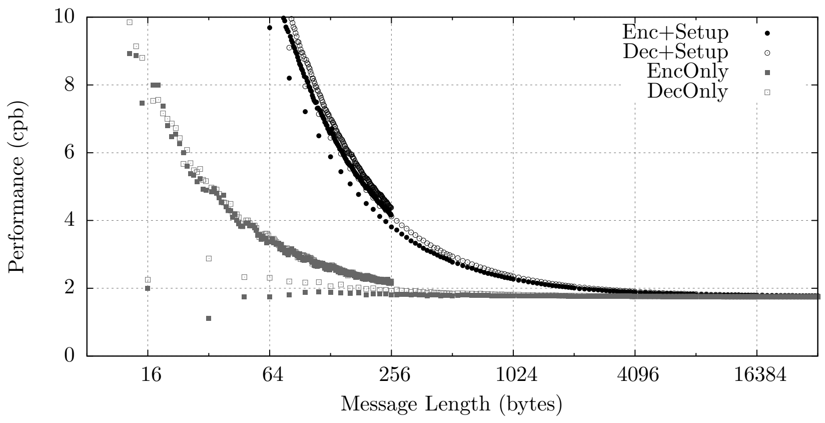


Figure 8.1.: Speed measurements in cycles per bytes for optimized implementations of POET-AES10-AES10 (top) and POET-AES10-AES4 (bottom) without intermediate tags on Haswell, when including the costs for key setup (Enc+setup/Dec+setup) or ignoring them (Enc only/dec only). The data and measuring description are given in Table 8.1 and the text.

mega128, with additional costs of about 1,100 gates. Table 8.2 summarizes the performance figures for the AES.

Platform	CPU	cpb	Ref.
64 bit	Intel i7-920 (Bloomfield)	6.92	[21]
	Intel Core 2 Q9550 (Yorkfield)	7.59	[21]
	Intel Core 2 Q6600 (Kentsfield)	9.32	[21]
32 bit	Intel Pentium 4 f12, x86 (Willamette)	14.13	[10]
	ARM Cortex-A8 (OpenSSL)	25.40	[37]
8 bit	Atmel AVR ATmega128 (extended)	78.70	[55]
	Atmel AVR AT90USB646	124.60	[40]

Table 8.2.: Speed of existing software implementations for one encryption of the AES, omitting the cost for the key schedule on selected common platforms; cpb = cycles per byte.

POET implementations without AES-NI. From the existing implementations of the AES, we can derive estimations for optimized implementations of POET, using the AES as a block cipher and four-round AES for universal hashing, on different platforms without NI. Therefore, we sum up the costs for 18 rounds of the AES. For either version, we add an overhead of one cpb for the chaining XOR operations.

The results by Käsper and Schwabe [21] provide an estimate that POET with four-round AES can run at about $1.8 \cdot 7 + 1 \approx 14$ cpb. Concerning 32-bit implementations, we can use the figures by Krovetz and Rogaway [37] to estimate the performance on an ARM Cortex-A8. Therefore, we approximate the costs for an AES encryption with 25.4 cpb from their figures for AES-CTR. Therefore, we have an upper bound of $1.8 \cdot 25.4 + 1 \approx 46$ cpb for POET with four-round AES for hashing on a mobile 32-bit CPU. However, there clearly exist various more powerful 32-bit CPUs. For 8-bit implementations, we expect POET to run at speeds of about 250 cpb on an off-the-shelf 8-bit Atmel AVR CPU.

Platform	CPU	cpb
64-bit (with AES-NI)	Intel Haswell	1.75
64-bit (without AES-NI)	Intel Bloomfield	≈ 14
32-bit (without AES-NI)	ARM Cortex-A8	≈ 46
8-bit	Atmel AVR ATmega128	≈ 250

Table 8.3.: Estimated speeds in cycles per byte of single-threaded implementations of POET POET-AES10-AES4 when processing a single message of ≥ 2048 bytes on common platforms.

Chapter 9

Design Rationale

Key Generation. The key generation follows the idea from [32]. The user supplies a k-bit secret key SK. The further keys are then generated by encrypting distinct constants using the AES under SK. Since the AES is a well-studied and secure block cipher (a secure PRP), we can ensure to obtain pairwise independent keys, which is crucial for our security analysis.

Three-Layer Structure. The basic idea behind POET's symmetric design is given by the secure XEX approach [46], where a block cipher is wrapped by XORing secret values. For POET, these values are given by X_{i-1} (top row) and Y_{i-1} (bottom row), which are given as output of F_{K_F} and F_{K_F}, i.e., two instances of the ϵ-AXU hash function family F using independent keys. Thus, we have shown that POET inherits the resistance against chosen-plaintext- and chosen-ciphertext-adversaries. Moreover, based on the chaining, we show that POET satisfies OCCA security (cf. Chapter 7.2), which on the other hand implies decryption misuse.

Tag Splitting. Since two of the application fields of POET are low-latency and restricted environments, we want to keep the overhead as small as possible. Therefore, POET inherits the provably secure tag-splitting approach from McOE [23], which provides length-preserving encryption/decryption.

Standard Primitives. Our recommended instances of POET are the AES as block cipher and four-round AES as hash function which lets users benefit from available native instruction sets of current processors. Since the AES is probably the best-studied, most widely-deployed ciphers nowadays, POET becomes easy to analyze.

Key Lengths. We recommend POE and POET for the use of 128-bit keys. It is straightforward to enhance POE and POET with a block cipher with longer keys, such as AES-256. However, the benefit of longer keys depends on the use case.

For attack settings where an adversary has unlimited oracle access, the security of POET is still upper bounded by the birthday bound of the state size. On the other hand, in settings where the oracle renews the key after $\ell \ll 2^{n/2}$ blocks were processed, POET may benefit from increased key lengths.

Header Processing. Since POET – beyond other things – is designed to provide high performance (without neglecting any security properties), we decided to use the fast, easily parallelizable, and provably secure PMAC(1) design [12, 46] to process the header. We considered also a variant of

PMAC1 wherein each but the final header block is processed by four-round AES. The major aspect that complicates the use of four-round AES is that the well-studied ϵ-AXU property of four-round AES would not suffice. Daemen and Rijmen [16] showed that the AES super-box possesses what they called plateau characteristics, i.e., differential characteristics that possess a probability of zero for many, but a probability of 2.5 times greater than the maximum expected differential probability for a subset of keys. Finding reasonably tight upper bounds for the maximum differential probability of four-round differentials of the AES still remains an interesting open problem. Therefore, we decided for the moment in favor of security of our header-processing step and refrained from replacing the full AES with a reduced-round version.

Absence of Hidden Weaknesses. We, the designers and submitters of POE and POET, declare that we have not hidden any weaknesses in this cipher.

Chapter 10

Acknowledgments

We thank Yu Sasaki and all reviewers of the FSE 2014 for their helpful comments. This work benefitted from the fruitful discussions at the Seminar on Symmetric Cryptography at Schloss Dagstuhl in January 2014, especially from the contributions of Daniel J. Bernstein and Tetsu Iwata. Furthermore, we would like to thank Jian Guo, Jérémy Jean, Thomas Peyrin, and Lei Wang who pointed out a mismatch between the specified and the analyzed version of POET in the pre-proceedings of the FSE 2014 [27]. We would like to thank Mridul Nandi 1) for pointing out a missing assumption in Theorem 7.1 2) revealing a security issue for POET-m with $m > 2$. So, POET-m has been removed from this paper. Finally, we thank Abdelraheem, Bogdanov, and Tischhauser [3] who applied the observations of Cid and Procter [44] to POET and thus, concretized the risk of weak keys when using a multiplication in $GF(2^{128})$ for universal hashing. Therefore, we abandoned the instance for POET with Galois-Field multiplication for hashing.

Chapter 11

Intellectual Property

To the best of our knowledge, neither POE, POET nor any of their instantiations is encumbered by any patents. We have not, and will not, apply for patents on any part of our design or anything in this document, and we are unaware of any other patents or patent filings that cover this work. The example source code is in the public domain and can be freely used. If any of this information changes, the submitters will promptly (and within at most one month) announce these changes on the `crypto-competitions` mailing list.

We make this submission to the CAESAR competition solely as individuals. Our respective employers neither endorse nor condemn this submission.

Chapter 12

Consent

The submitters hereby consent to all decisions of the CAESAR selection committee regarding the selection or non-selection of this submission as a second-round candidate, a third-round candidate, a finalist, a member of the final portfolio, or any other designation provided by the committee. The submitters understand that the committee will not comment on the algorithms, except that for each selected algorithm the committee will simply cite the previously published analyses that led to the selection of the algorithm. The submitters understand that the selection of some algorithms is not a negative comment regarding other algorithms, and that an excellent algorithm might fail to be selected simply because not enough analysis was available at the time of the committee decision. The submitters acknowledge that the committee decisions reflect the collective expert judgments of the committee members and are not subject to appeal. The submitters understand that if they disagree with published analyses then they are expected to promptly and publicly respond to those analyses, not to wait for subsequent committee decisions. The submitters understand that this statement is required as a condition of consideration of this submission by the CAESAR selection committee.

Bibliography

[1] D. Eastlake 3rd. Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH). RFC 4305 (Proposed Standard), December 2005. Obsoleted by RFC 4835.

[2] Mohamed Ahmed Abdelraheem, Peter Beelen, Andrey Bogdanov, and Elmar Tischhauser. Twisted Polynomials and Forgery Attacks on GCM. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 762–786. Springer, 2015.

[3] Mohamed Ahmed Abdelraheem, Andrey Bogdanov, and Elmar Tischhauser. Weak-Key Analysis of POET. Cryptology ePrint Archive, Report 2014/226, 2014. `http://eprint.iacr.org/`.

[4] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to Securely Release Unverified Plaintext in Authenticated Encryption. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, pages 105–125, 2014.

[5] Mihir Bellare, Alexandra Boldyreva, Lars R. Knudsen, and Chanathip Namprempre. On-line Ciphers and the Hash-CBC Constructions. *Journal of Cryptology*, 25(4):640–679, 2012.

[6] Mihir Bellare and Chanathip Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000.

[7] Mihir Bellare and Phillip Rogaway. Encode-Then-Encipher Encryption: How to Exploit Nonces or Redundancy in Plaintexts for Efficient Cryptography. In *ASIACRYPT*, pages 317–330, 2000.

[8] Mihir Bellare and Phillip Rogaway. Code-Based Game-Playing Proofs and the Security of Triple Encryption. *IACR Cryptology ePrint Archive*, 2004:331, 2004.

[9] Mihir Bellare, Phillip Rogaway, and David Wagner. The EAX Mode of Operation. In Bimal K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 389–407. Springer, 2004.

[10] Daniel J. Bernstein and Peter Schwabe. New AES Software Speed Records. In *INDOCRYPT*, pages 322–336, 2008.

[11] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography*, volume 7118 of *Lecture Notes in Computer*

Science, pages 320–337. Springer, 2011.

[12] John Black and Phillip Rogaway. A Block-Cipher Mode of Operation for Parallelizable Message Authentication. In *EUROCRYPT*, pages 384–397, 2002.

[13] Martin Boesgaard, Thomas Christensen, and Erik Zenner. Badger - A Fast and Provably Secure MAC. In John Ioannidis, Angelos D. Keromytis, and Moti Yung, editors, *ACNS*, volume 3531 of *Lecture Notes in Computer Science*, pages 176–191, 2005.

[14] Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting Mobile Communications: The Insecurity of 802.11. In *MOBICOM*, pages 180–189, 2001.

[15] Larry Carter and Mark N. Wegman. Universal Classes of Hash Functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.

[16] Joan Daemen and Vincent Rijmen. Plateau Characteristics. *IET Information Security*, 1(1):11–17, 2007.

[17] Nilanjan Datta and Mridul Nandi. ELmD. `http://competitions.cr.yp.to/caesar-submissions.html`, 2014.

[18] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard), January 1999. Obsoleted by RFC 4346, updated by RFCs 3546, 5746, 6176.

[19] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346 (Proposed Standard), April 2006. Obsoleted by RFC 5246, updated by RFCs 4366, 4680, 4681, 5746, 6176.

[20] 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.

[21] Emilia Käsper and Peter Schwabe. Faster and Timing-Attack Resistant AES-GCM. In *CHES*, pages 1–17, 2009.

[22] Ewan Fleischmann, Christian Forler, and Stefan Lucks. McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. In *FSE*, pages 196–215, 2012.

[23] Ewan Fleischmann, Christian Forler, Stefan Lucks, and Jakob Wenzel. McOE: A Foolproof On-Line Authenticated Encryption Scheme. Cryptology ePrint Archive, Report 2011/644, 2011. `http://eprint.iacr.org/`.

[24] Agner Fog. Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs, Aug 07 2014. `http://www.agner.org/optimize/instruction_tables.pdf`.

[25] Pierre-Alain Fouque, Antoine Joux, Gwenaëlle Martinet, and Frédéric Valette. Authenticated On-Line Encryption. In *Selected Areas in Cryptography*, pages 145–159, 2003.

[26] Shay Gueron. Intel® Advanced Encryption Standard (AES) Instructions Set - Rev 3.01. Intel White Paper. Technical report, Intel corporation, September 2012.

[27] Jian Guo, Jérémy Jean, Thomas Peyrin, and Wang Lei. Breaking POET Authentication with a Single Query. Cryptology ePrint Archive, Report 2014/197, 2014. `http://eprint.iacr.org/`.

[28] Viet Tung Hoang, Reza Reyhanitabar, Phillip Rogaway, and Damian Vizár. Online Authenticated-Encryption and its Nonce-Reuse Misuse-Resistance. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 493–517. Springer, 2015.

[29] Viet Tung Hoang, Reza Reyhanitabar, Phillip Rogaway, and Damian Vizár. Online Authenticated-Encryption and its Nonce-Reuse Misuse-Resistance. Cryptology ePrint Archive, Report 2015/189, 2015. `http://eprint.iacr.org/`.

[30] George Hotz. Console Hacking 2010 - PS3 Epic Fail. 27th Chaos Communications Congress, 2010. `http://tinyurl.com/39yds8h`.

[31] ITU-T. Interfaces for the Optical Transport Network (OTN). Recommendation G.709/Y.1331,

International Telecommunication Union, Geneva, December 2009.

[32] Tetsu Iwata and Kaoru Kurosawa. OMAC: One-Key CBC MAC. In Thomas Johansson, editor, *FSE*, volume 2887 of *Lecture Notes in Computer Science*, pages 129–153. Springer, 2003.

[33] Jonathan Katz and Moti Yung. Unforgeable Encryption and Chosen Ciphertext Secure Modes of Operation. In *FSE*, pages 284–299, 2000.

[34] Liam Keliher and Jiayuan Sui. Exact Maximum Expected Differential and Linear Probability for Two-Round Advanced Encryption Standard. *IET Information Security*, 1(2):53–57, 2007.

[35] S. Kent. Ip encapsulating security payload (esp). RFC 4303 (Proposed Standard), December 2005.

[36] Tadayoshi Kohno. Attacking and Repairing the WinZip Encryption Scheme. In *ACM Conference on Computer and Communications Security*, pages 72–81, 2004.

[37] Ted Krovetz and Phillip Rogaway. The Software Performance of Authenticated-Encryption Modes. In *FSE*, pages 306–327, 2011.

[38] Raymond Manley and David Gregg. A Program Generator for Intel AES-NI Instructions. In Guang Gong and Kishan Chand Gupta, editors, *Progress in Cryptology - INDOCRYPT 2010 - 11th International Conference on Cryptology in India, Hyderabad, India, December 12-15, 2010. Proceedings*, volume 6498 of *Lecture Notes in Computer Science*, pages 311–327. Springer, 2010.

[39] David McGrew and John Viega. The Galois/Counter Mode of Operation (GCM). *Submission to NIST. `http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/gcm/gcm-spec.pdf`*, 2004.

[40] Dag Arne Osvik, Joppe W. Bos, Deian Stefan, and David Canright. Fast Software AES Encryption. In *FSE*, pages 75–93, 2010.

[41] J. Postel. User Datagram Protocol. RFC 768 (INTERNET STANDARD), August 1980.

[42] J. Postel. Internet Protocol. RFC 791 (INTERNET STANDARD), September 1981. Updated by RFCs 1349, 2474, 6864.

[43] J. Postel. Transmission Control Protocol. RFC 793 (INTERNET STANDARD), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[44] Gordon Procter and Carlos Cid. On Weak Keys and Forgery Attacks against Polynomial-based MAC Schemes. In *Fast Software Encryption, 20th International Workshop, FSE 2013*, Lecture Notes in Computer Science - LNCS. Springer, 2013.

[45] Phillip Rogaway. Authenticated-Encryption with Associated-Data. In *ACM Conference on Computer and Communications Security*, pages 98–107, 2002.

[46] Phillip Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004.

[47] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption. In *ACM Conference on Computer and Communications Security*, pages 196–205, 2001.

[48] Phillip Rogaway and Thomas Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 373–390. Springer, 2006.

[49] Phillip Rogaway and Thomas Shrimpton. Deterministic Authenticated-Encryption: A Provable-Security Treatment of the Key-Wrap Problem. Cryptology ePrint Archive, Report 2006/221. (Full Version), 2006. `http://eprint.iacr.org/`.

[50] Phillip Rogaway and Haibin Zhang. Online Ciphers from Tweakable Blockciphers. In Aggelos

Kiayias, editor, *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, volume 6558 of *Lecture Notes in Computer Science*, pages 237–249. Springer, 2011.

[51] Markku-Juhani Olavi Saarinen. Cycling Attacks on GCM, GHASH and Other Polynomial MACs and Hashes. In Anne Canteaut, editor, *FSE*, volume 7549 of *Lecture Notes in Computer Science*, pages 216–225. Springer, 2012.

[52] Todd Sabin. Vulnerability in Windows NT's SYSKEY encryption. *BindView Security Advisory*, 1999. Available at `http://marc.info/?l=ntbugtraq&m=94537191024690&w=4`.

[53] Douglas R. Stinson. Universal Hashing and Authentication Codes. In Joan Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 74–85. Springer, 1991.

[54] Douglas R. Stinson. Universal Hashing and Authentication Codes. *Des. Codes Cryptography*, 4(4):369–380, 1994.

[55] Stefan Tillich and Christoph Herbst. Boosting AES Performance on a Tiny Processor Core. In *CT-RSA*, pages 170–186, 2008.

[56] Mark N. Wegman and J. Lawrence Carter. New Hash Functions and Their Use in Authentication and Set Equality. *Journal of Computer and System Sciences*, 22(3):265–279, June 1981.

[57] Bo Zhu, Yin Tan, and Guang Gong. Revisiting MAC Forgeries, Weak Keys and Provable Security of Galois/Counter Mode of Operation. In *Cryptology and Network Security*, pages 20–38. Springer, 2013.

Appendix A

Lemmas of the OPERMCCA Analysis of POE

A.1. Upper Bound for $\mathsf{COLL}^{\mathsf{enc}}$

Lemma A.1 ($\mathbf{COLL}^{\mathbf{enc}}$). *Let M_i, M'_j denote the i-th and j-th block of one or two encryption queries $M, M' \in \mathcal{Q}$, and X_i, X'_j the internal top-row chaining values as defined in Algorithm 6. Let COLL^{enc} be the event that $X_i = X'_j$ for two distinct tuples (X_{i-1}, M_i) and (X'_{j-1}, M'_j), with $i, j \geq 1$, as defined in Section 7.1. Then, it holds that*

$$\Pr\left[\mathsf{COLL}^{enc}\right] \leq \frac{\ell(\ell+1)\epsilon}{2}.$$

Proof. The adversary has full control over the message blocks M_i and M'_j, which can refer to blocks in two messages as well as to different blocks in the same message. In encryption direction, the adversary never sees the values X_i and X'_j that serve as input to the encryption layer.

In the following, we study the difference in the behavior of POE and P for two mutually exclusive cases and derive the advantage for each of them.

Case (1): $X_{i-1} = X'_{j-1}$**.** This case can happen when M and M' share a common prefix up to the $(i-1)$-th message block; otherwise, $\mathcal{A}$ would have already found a collision at this point and we would have given the attack to the adversary before. Hence, the advantage for $\mathcal{A}$ is 0 in the latter case. In the former case, from $(X_{i-1}, M_i) \neq (X'_{j-1}, M'_j)$ must follow that $M_i \neq M'_j$. Since X_i and X'_j are computed by $F_{K_F}(X_{i-1}) \oplus M_i$ and $F_{K_F}(X'_{j-1}) \oplus M'_j$, it must hold for a collision between X_i and X'_j

$$F_{K_F}(X_{i-1}) \oplus M_i = F_{K_F}(X'_{j-1}) \oplus M'_j,$$

with $F_{K_F}(X_{i-1}) = F_{K_F}(X'_{j-1})$. It is trivial to see that this condition can never hold, and the advantage for $\mathcal{A}$ is 0 in this case.

Case (2): $X_{i-1} \neq X'_{j-1}$**.** Since $F_{K_F}(\cdot)$ is an ϵ-AXU family of hash functions, we can derive a family of hash functions $F'_{K_F}(\cdot, \cdot)$ as

$$F'_{K_F}(x, m) := F_{K_F}(x) \oplus m,$$

which is ϵ-AU according to Theorem 4.3. For a collision of the form $X_i = X'_j$, it must hold that

$$F'_{K_F}(X_{i-1}, M_i) = F'_{K_F}(X'_{j-1}, M'_j).$$

for distinct inputs (X_{i-1}, M_i) and (X'_{j-1}, M'_j). Since the adversary queries at most ℓ blocks, there are at most $\ell(\ell-1)/2$ options for two-block collisions among them. Moreover, each of the chaining values could also collide with X_0 (which is a constant for POE) with probability at most ϵ. Hence, the probability of $\mathsf{COLL}^{\text{enc}}$ to happen can be upper bounded by

$$\Pr\left[\mathsf{COLL}^{\text{enc}}\right] \leq \frac{\ell(\ell-1)\epsilon}{2} + \ell\epsilon \leq \frac{\ell(\ell+1)\epsilon}{2}.$$

□

A.2. Upper Bound for NOCOLLWIN

Lemma A.2 (NOCOLLWIN). *Let NOCOLLWIN be the event as defined in Equation (7.5). Then, it holds that*

$$\Pr[\mathit{NOCOLLWIN}] \leq \frac{\ell^2}{2^n - \ell}.$$

Proof. Here, we regard the case that $\mathcal{A}$ shall distinguish $\mathsf{POE}_{\pi,F}$, $\mathsf{POE}^{-1}_{\pi^{-1},F}$ from a random on-line permutation P, P^{-1} when no internal collisions occur. Prior, we generalize that each distinct query pair (M, C), $(M', C') \in \mathcal{Q}$ shares a common prefix of i blocks. We denote the by i the length of their longest common prefix:

$$i = \mathsf{LLCP}_n(M, M') = \max_h \left\{\forall j \in 0, \ldots, h : M_j = M'_j\right\}.$$

In the following, we study the difference in the behavior of POE and P for three subcases, and derive the advantage of $\mathcal{A}$ to distinguish between POE and P for each of them.

Case (1): Message Blocks in the Common Prefix. The input and output behaviors of $(\mathsf{POE}_{\pi,F}, \mathsf{POE}^{-1}_{\pi^{-1},F})$ and (P, P^{-1}) are identical for the common prefix. Hence, the advantage for $\mathcal{A}$ to distinguish between them is 0 in this case.

Case (2): Message Block Directly after the Common Prefix. Since M and M' share a common prefix of i blocks, it must hold that $X_i = X'_i$ and $Y_i = Y'_i$. For an encryption query with the inputs $M_{i+1} \neq M'_{i+1}$, it must hold that

$$X_{i+1} = F_{K_F}(X_i) \oplus M_{i+1} \neq F_{K_F}(X'_i) \oplus M'_{i+1} = X'_{i+1}.$$

Since π is a permutation, it must follow that $C_{i+1} \neq C'_{i+1}$:

$$C_{i+1} = F_{K_F}(Y_i) \oplus \pi(X_{i+1}) \neq F_{K_F}(Y'_i) \oplus \pi(X'_{i+1}) = C'_{i+1}.$$

The analysis is similar in decryption direction. In the random case, P or P^{-1} are used with two different prefixes $(M_1 \mid\mid \ldots \mid\mid M_{i+1})$ and $(M'_1 \mid\mid \ldots \mid\mid M'_{i+1})$ in encryption, or $(C_1 \mid\mid \ldots \mid\mid C_{i+1})$ and $(C'_1 \mid\mid \ldots \mid\mid C'_{i+1})$ in decryption direction. Since P and P^{-1} are random on-line permutations, $C_{i+1} \neq C'_{i+1}$ or $M_{i+1} \neq M'_{i+1}$ must also hold in this case, respectively. Hence, the behavior of $(\mathsf{POE}_{\pi,F}, \mathsf{POE}^{-1}_{\pi^{-1},F})$ and a random on-line permutation is also identical for the message block after the common prefix, and the advantage for $\mathcal{A}$ to distinguish them is 0 also in this case.

Case (3): After the $(i+1)$-th Message Block. In the random case, each query output is chosen uniformly at random from the set $\{0,1\}^n$. However, in the real world, each output of either an encryption or a decryption query is chosen uniformly at random from the set $\{0,1\}^n \setminus \mathcal{Q}$. This means that in the real case, POE loses randomness with every block of every query. Therefore, we can upper bound the probability of $\mathcal{A}$ to distinguish POE from a random on-line permutation by

$$\frac{\ell^2}{2^n - \ell}.$$

Our claim follows from summing up the individual terms. □

Appendix B

Lemmas of the OCCA Analysis of POET

B.1. Upper Bound for $\mathsf{COLL}^{\mathsf{enc}}$

Corollary B.1 ($\mathbf{COLL}^{\mathbf{enc}}$). *Let M_i, M'_j denote the i-th and j-th block of one or two encryption queries $M, M' \in \mathcal{Q}$, and X_i, X'_j the internal top-row chaining values as defined in Algorithm 6. Let COLL^{enc} be the event that $X_i = X'_j$ for two distinct tuples (X_{i-1}, M_i) and (X'_{j-1}, M'_j), with $i, j \geq 1$, as defined in Section 7.2. Then, it holds that*

$$\Pr\left[\mathsf{COLL}^{enc}\right] \leq \frac{(\ell + q)^2}{2}\epsilon.$$

Proof. Since the structures of POE and POET are inherently similar, the proof follows the same argumentation as that for Lemma A.1. Again, the adversary has full control over the message blocks M_i and M'_j, which can refer to blocks in two messages as well as to different blocks in the same message. In encryption direction, the adversary never sees the values X_i and X'_j that serve as input to the encryption layer. We can distinguish between the same two cases.

Case (1): $X_{i-1} = X'_{j-1}$**.** As in Case (1) of the proof of Lemma A.1, the advantage for $\mathcal{A}$ is 0 in this case.

Case (2): $X_{i-1} \neq X'_{j-1}$**.** Again, the probability for two fixed top-row chaining values to collide can be upper bounded by ϵ. Though, for POET, the initial chaining values X_0 are not constant, but are the result of the **ProcessHeader** step. Therefore, there are at most $\ell + q$ possible chaining values when $\mathcal{A}$ poses q queries of ℓ blocks. So, the probability of $\mathsf{COLL}^{\mathrm{enc}}$ can be upper bounded for POET by

$$\Pr\left[\mathsf{COLL}^{\mathrm{enc}}\right] \leq \frac{(\ell + q)^2}{2}\epsilon.$$

The bound in Corollary B.1 follows. □

B.2. Upper Bound for $\mathsf{COLL}^{\mathsf{ad}}$

Since the header-processing step in POET is similar to PMAC1, we recall the security bound from Corollary 17 in [46].

Theorem B.2 (Security of PMAC1 ([46])). *Let $\mathcal{A}$ be a PRF adversary which runs in time at most $O(t)$, and asks at most q queries of a total length of at most ℓ n-bit blocks to an oracle which uses either PMAC1 instantiated with an ideal n-bit permutation π or a uniformly at random chosen PRF $\Gamma : \{0,1\}^* \to \{0,1\}^n$. Then, the maximal advantage of $\mathcal{A}$ to distinguish between $PMAC1[\pi]$ and Γ is upper bounded by*

$$\mathbf{Adv}^{\mathit{PRF}}_{PMAC1[\pi]}(\mathcal{A}) \leq \frac{5.5\ell^2}{2^n},$$

where π is chosen uniformly at random from the set of all n-bit permutations.

Since POET adds an additional block to each header for encoding ℓ_s and ℓ_t and a second additional block if the length of the header is a multiple of the block length, we can derive from Theorem B.2 the following corollary.

Corollary B.3 ($\mathbf{COLL}^{\mathbf{ad}}$). *Let H and H' denote two distinct headers. Let COLL^{ad} be the event that **ProcessHeader**(H) = **ProcessHeader**(H') for two distinct headers H and H', as defined in Section 7.2. Then, it holds that*

$$\Pr\left[\mathsf{COLL}^{ad}\right] \leq \frac{5.5(\ell+2q)^2}{2^n}.$$

B.3. Upper Bound for $\mathsf{COLL}^{\mathsf{lmb}}$

The encryption and decryption procedures of POET differ from those of POE only in the way how POET treats the last message block. To prove the OCCA security of POET, we have to consider the probability for collisions of internal chaining values when M_i or M'_j (or both) refer to the last blocks of M and M', respectively.

Lemma B.4 ($\mathbf{COLL}^{\mathbf{lmb}}$). *Let M_i, M'_j denote the i-th and j-th block of one or two encryption queries $M, M' \in \mathcal{Q}$, and X_i, X'_j the internal top-row chaining values as defined in Algorithm 6. Further, let m denote the number of n-bit blocks in M and m' the number of n-bit blocks in M'. Let COLL^{lmb} be the event that $X_i = X'_j$ for two distinct tuples (X_{i-1}, M_i) and (X'_{j-1}, M'_j), with $i, j \geq 1$ and $i = m$, as defined in Section 7.2. Then, it holds that*

$$\Pr\left[\mathsf{COLL}^{lmb}\right] \leq \max\left\{q(\ell+q)\epsilon, \frac{q(\ell+q)}{2^n - q}\right\}.$$

Proof. In the following, we want to upper bound the probability of a collision in the chaining values $X_i = X'_j$ for two distinct tuples (X_{i-1}, M_i) and (X'_{j-1}, M'_j), with $i, j \geq 1$. Wlog., we define that M_i is the last block of M, i.e., $i = m$, but the argumentation would be similar if M_i was the last block of M'. At the end, we consider a third possible option: that the last block of a message collidse with one of the initial chaining values X_0.

POET pads a message whose length is not a multiple of n with the most significant bits of the result from the header-processing step, τ: $M_m \mid\mid \tau^\alpha$. Hence, we have to analyze two cases: (1) that M_m is either a full block or (2) that M_m is padded.

In each case, we regard two subcases: $X_{i-1} = X'_{j-1}$ and $X_{i-1} \neq X'_{j-1}$, where $X_{i-1} = X'_{j-1}$ may result either from a common prefix of M and M', or from a previous collision that $\mathcal{A}$ found before. Since we would have already given the attack to $\mathcal{A}$ in the latter case (and its advantage would be 0 then), we can limit our consideration to when $X_{i-1} = X'_{j-1}$ was due to a common prefix. Note that $(X_{i-1}, M_i) \neq (X'_{j-1}, M'_j)$ implies that $M_i \neq M'_j$; otherwise, M_i and M'_j would just extend the common prefix and the advantage for $\mathcal{A}$ would be 0 again.

For each subcase, we analyze three mutually exclusive constellations, depending on M'_j:

1. M'_j is an intermediate block of M': $j < m'$.
2. M'_j is the last, unpadded block of M': $j = m'$ and $|M'_j| = n$.
3. M'_j is the last, padded block of M': $j = m'$ and $|M'_j| < n$.

We will derive the advantage for each subcase separately.

Case (1): Without Tag-Splitting at M_m. Here, the top-row chaining value X_m is computed by $F_{K_F}(X_{m-1}) \oplus M_m \oplus S$. Depending on the constellations mentioned above, X'_j is computed slightly differently. For a collision of the form $X_m = X'_j$, it must hold

$$F_{K_F}(X_{m-1}) \oplus M_m \oplus S = \begin{cases} F_{K_F}(X'_{j-1}) \oplus M'_j & \text{if } j < m', \\ F_{K_F}(X'_{m'-1}) \oplus M'_{m'} \oplus S' & \text{if } j = m', |M'_{m'}| = n, \\ F_{K_F}(X'_{m'-1}) \oplus (M'_{m'} \mid\mid {\tau'}^{\alpha}) \oplus S' & \text{if } j = m', |M'_{m'}| < n. \end{cases}$$

These cover all possible cases for a collision in X_m and X'_j when M_m is not padded. To simplify our analysis, we make $\mathcal{A}$ stronger than it is over Case (1) and give it full control over ${\tau'}^{\alpha}$.

Subcase (1.1): $X_{m-1} = X'_{j-1}$. Since $F_{K_F}(\cdot)$ is a function, it must hold that $F_{K_F}(X_{m-1}) = F_{K_F}(X_{j-1})$. Hence, we can rearrange our equations from above and see that a collision in X_m and X'_j requires that $\mathcal{A}$ must choose M_m and M'_j such that

$$\begin{aligned} M_m \oplus M'_j &= S \text{ or} \\ M_m \oplus M'_{m'} &= S \oplus S' \text{ or} \\ M_m \oplus (M'_{m'} \mid\mid {\tau'}^{\alpha}) &= S \oplus S' \end{aligned}$$

holds. Since M_m and M'_j must differ in this case, trivial collisions of the form $S = S'$ are ruled out. Since S and S' are secret, the success probability for $\mathcal{A}$ is at most $\frac{1}{2^n - q}$.

Subcase (1.2): $X_{m-1} \neq X'_{j-1}$. This time, we can rearrange our equations from above and see that a collision in X_m and X'_j requires that $\mathcal{A}$ must find

$$\begin{aligned} F_{K_F}(X_{m-1}) \oplus F_{K_F}(X'_{j-1}) &= M_m \oplus M'_j \oplus S \text{ or} \\ F_{K_F}(X_{m-1}) \oplus F_{K_F}(X'_{m'-1}) &= M_m \oplus M'_{m'} \oplus S \oplus S' \text{ or} \\ F_{K_F}(X_{m-1}) \oplus F_{K_F}(X'_{m'-1}) &= M_m \oplus (M'_{m'} \mid\mid {\tau'}^{\alpha}) \oplus S \oplus S'. \end{aligned}$$

Since $F_{K_F}(\cdot)$ is an ϵ-AXU family of hash functions, the success probability that $\mathcal{A}$ can choose M_m and M'_j appropriately can be upper bounded by ϵ for any of these constellations.

In both subcases, M_m can be any of the q last message blocks and M'_j any of the ℓ blocks of all queries. So, the success probability of $\mathcal{A}$ for Case (1) can be upper bounded by

$$\max\left\{\ell q\epsilon, \frac{\ell q}{2^n - q}\right\}.$$

Case (2): With Tag-Splitting at M_m. Now, the top-row chaining value X_m is computed by $F_{K_F}(X_{m-1}) \oplus (M_m \mid\mid \tau^\alpha) \oplus S$. For a collision of the form $X_m = X'_j$, it must hold that

$$X_m = \begin{cases} F_{K_F}(X'_{j-1}) \oplus M'_j & \text{if } j < m', \\ F_{K_F}(X'_{m'-1}) \oplus M'_{m'} \oplus S' & \text{if } j = m', |M'_{m'}| = n, \\ F_{K_F}(X'_{m'-1}) \oplus (M'_{m'} \mid\mid \tau'^\alpha) \oplus S' & \text{if } j = m', |M'_{m'}| < n. \end{cases}$$

Again, these cover all possible constellations for a collision in X_m and X'_j when M_m is padded. For the sake of simplicity, we make $\mathcal{A}$ again stronger than it is and give it full control over τ^α and τ'^α in the following.

Subcase (2.1): $X_{m-1} = X'_{j-1}$. Clearly, in this subcase $F_{K_F}(X_{m-1}) = F_{K_F}(X'_{j-1})$ must hold. Hence, we can rearrange our equations from above and see that a collision in X_m and X'_j requires that $\mathcal{A}$ must find

$$\begin{gathered} (M_m \mid\mid \tau^\alpha) \oplus M'_j = S \text{ or} \\ (M_m \mid\mid \tau^\alpha) \oplus M'_{m'} = S \oplus S' \text{ or} \\ (M_m \mid\mid \tau^\alpha) \oplus (M'_{m'} \mid\mid \tau'^\alpha) = S \oplus S'. \end{gathered}$$

In all constellations, $\mathcal{A}$ has to choose M_m and M'_j appropriately to match S or $S \oplus S'$. Furthermore, since this subcase implies $M_m \neq M'_j$, trivial collision for $S = S'$ are ruled out again. Since S and S' are secret, the success probability for $\mathcal{A}$ in all three constellations is at most $\frac{1}{2^n-q}$.

Subcase (2.2): $X_{m-1} \neq X'_{j-1}$. Again, we can rearrange our equations from above and see that a collision in X_m and X'_j requires that $\mathcal{A}$ must find

$$\begin{gathered} F_{K_F}(X_{m-1}) \oplus F_{K_F}(X'_{j-1}) = (M_m \mid\mid \tau^\alpha) \oplus M'_j \oplus S \text{ or} \\ F_{K_F}(X_{m-1}) \oplus F_{K_F}(X'_{m'-1}) = (M_m \mid\mid \tau^\alpha) \oplus M'_{m'} \oplus S \oplus S' \text{ or} \\ F_{K_F}(X_{m-1}) \oplus F_{K_F}(X'_{m'-1}) = (M_m \mid\mid \tau^\alpha) \oplus (M'_{m'} \mid\mid \tau'^\alpha) \oplus S \oplus S'. \end{gathered}$$

Since $F_{K_F}(\cdot)$ is an ϵ-AXU family of hash functions, the success probability that $\mathcal{A}$ can choose M_m and M'_j appropriately can be upper bounded by ϵ in either constellation.
Similar to Case (1), we can upper bound the success probability of $\mathcal{A}$, asking at most q queries of a total length of ℓ blocks, for Case (2) by

$$\max\left\{\ell q\epsilon, \frac{\ell q}{2^n - q}\right\}.$$

Case (3): Collision with Initial Chaining Value. For a collision of the form $X_m = X'_0$, it must hold that

$$X'_0 = \begin{cases} F_{K_F}(X_{m-1}) \oplus M_m \oplus S & \text{if } j = m', |M'_{m'}| = n, \\ F_{K_F}(X_{m-1}) \oplus (M_m \mid\mid \tau^\alpha) \oplus S & \text{if } j = m', |M_m| < n. \end{cases}$$

These cover all possible cases for a collision in X_m and X'_0. To simplify our analysis, we make $\mathcal{A}$ stronger than it is and give it full control over τ^{α}.

Since the initial chaining values $X'_0 = \tau'$ and encrypted message lengths S are secret, the adversary must choose M_m appropriately. Since $F'_{K_F}(x, y) := F_{K_F}(x) \oplus y$ is an ϵ-AU family of hash functions, it follows that the probability for a collision with fixed X'_0 and X_m is upper bounded by ϵ. Over q queries with q initial chaining values and q last message blocks, the probability is at most $q^2\epsilon$.

Since Case (1) and Case (2) are mutually exclusive, the success probability of $\mathcal{A}$ for the event $\mathsf{COLL}^{\mathrm{lmb}}$ is given by the maximum of the success probabilities of the two cases plus that for Case (3). Thus, it holds that

$$\Pr\left[\mathsf{COLL}^{\mathrm{lmb}}\right] \le \max\left\{q\ell\epsilon, \frac{q\ell}{2^n - q}\right\} + q^2\epsilon \le \max\left\{q(\ell+q)\epsilon, \frac{q(\ell+q)}{2^n - q}\right\}.$$

□

Appendix C

Test Vectors for **POET**

C.1. Four-Round AES

SK:	`01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10`	(16 octets)
K:	`db f1 84 11 2e b9 11 16 59 71 2b af cf f2 ab 24`	(16 octets)
L:	`9a 7a 06 19 aa c2 9e 6c 1f 2b 5c 47 53 d5 88 f3`	(16 octets)
K_F:	`14 2c 51 c9 af 2c f1 d9 2e 89 37 c4 fb c1 8d 7a`	(16 octets)

H:		(0 octets)
τ:	`65 32 57 29 54 d6 7d be 75 22 11 b9 c1 1c 56 07`	(16 octets)
M:	`00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff`	(16 octets)
C:	`de 79 29 b3 a8 28 8f 48 93 1e b3 97 4b 40 ad 60`	(16 octets)
T:	`40 13 1a be 5d d7 a3 1f 99 72 92 20 f1 33 eb 1e`	(16 octets)

H:	`00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff` `de ad be ef de af ba be`	(24 octets)
τ:	`31 23 44 18 68 e4 49 4f 9b 15 7d b4 bb a5 7f 75`	(16 octets)
M:		(0 octets)
C:		(0 octets)
T:	`42 c5 a1 70 cf 74 5b b8 ce 84 51 ad 83 38 d7 27`	(16 octets)

SK:	00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff	(16 octets)
K:	fd e4 fb ae 4a 09 e0 20 ef f7 22 96 9f 83 83 2b	(16 octets)
L:	84 d4 c9 c0 8b 4f 48 28 61 e3 a9 c6 c3 5b c4 d9	(16 octets)
K_F:	1d f9 27 37 45 13 bf d4 9f 43 6b d7 3f 32 52 85	(16 octets)
H:	01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10	(16 octets)
τ:	7d 04 02 ee 3e f4 06 5c 7b ed 3a ff 96 84 df b7	(16 octets)

M:	00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f	(56 octets)
	10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f	
	20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f	
	de ad be ef de af ba be	
C:	bc 0c da 83 a5 a4 7b 1f 3d 06 3e 07 a7 10 ba d0	(56 octets)
	6f 0e 93 f8 b5 12 32 c7 ef a4 dd 45 6a dc dc 92	
	af 48 ec 20 59 9f 05 cb 0f 1e 4e 22 11 be eb 47	
	b8 ca 69 f0 d1 a4 7e cb	
T:	50 dc ff 4d b0 51 99 48 d7 1f 5c 95 43 ad 43 2e	(16 octets)

H:	00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff	(24 octets)
	de ad be ef de af ba be	
τ:	17 99 c6 a8 df 70 85 17 23 04 cd 46 46 39 41 3c	(16 octets)
M:	00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f	(52 octets)
	10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f	
	20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f	
	fe fe ba be	
C:	67 f1 28 b1 63 10 cd 0c cd 3b c3 c3 07 a5 00 18	(52 octets)
	fc 4c 31 73 30 9b 5a 7b eb a1 05 95 47 ba 31 3c	
	34 64 d9 b3 fb 48 8b 79 89 b8 87 5e 55 d9 a9 43	
	81 a0 7d 2b	
T:	34 0b 40 c2 ae 24 34 79 d7 a2 f4 e5 b5 0d b8 20	(16 octets)

C.2. Full-Round AES

SK:	`01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10`	(16 octets)
K:	`db f1 84 11 2e b9 11 16 59 71 2b af cf f2 ab 24`	(16 octets)
L:	`9a 7a 06 19 aa c2 9e 6c 1f 2b 5c 47 53 d5 88 f3`	(16 octets)
K_F:	`14 2c 51 c9 af 2c f1 d9 2e 89 37 c4 fb c1 8d 7a`	(16 octets)

H:		(0 octets)
τ:	`65 32 57 29 54 d6 7d be 75 22 11 b9 c1 1c 56 07`	(16 octets)
M:	`00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff`	(16 octets)
C:	`7a 15 53 d4 14 78 b2 99 3a 4c 19 70 d2 41 04 56`	(16 octets)
T:	`df 9e eb 7e 56 61 a7 8f 72 93 a1 f4 50 ab 71 37`	(16 octets)

H:	`00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff`	(24 octets)
	`de ad be ef de af ba be`	
τ:	`31 23 44 18 68 e4 49 4f 9b 15 7d b4 bb a5 7f 75`	(16 octets)
M:		(0 octets)
C:		(0 octets)
T:	`51 ad 44 5b 59 ca bb 77 9e cc 29 8e 18 3e 36 7a`	(16 octets)

SK:	`00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff`	(16 octets)
K:	`fd e4 fb ae 4a 09 e0 20 ef f7 22 96 9f 83 83 2b`	(16 octets)
L:	`84 d4 c9 c0 8b 4f 48 28 61 e3 a9 c6 c3 5b c4 d9`	(16 octets)
K_F:	`1d f9 27 37 45 13 bf d4 9f 43 6b d7 3f 32 52 85`	(16 octets)

H:	`01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10`	(16 octets)
τ:	`7d 04 02 ee 3e f4 06 5c 7b ed 3a ff 96 84 df b7`	(16 octets)
M:	`00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f`	(56 octets)
	`10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f`	
	`20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f`	
	`de ad be ef de af ba be`	
C:	`4b 43 0f 48 03 ae 40 ea a8 95 42 bd 44 70 81 80`	(56 octets)
	`46 07 d1 57 7a c0 fd 90 a0 b0 53 a4 ea 4f c7 66`	
	`d8 d6 38 4e 83 fa bc 26 5d be ee 32 6f b1 0c 9e`	
	`9e 63 c1 e0 79 22 8b d5`	
T:	`67 5e fa 65 08 ea 2e f3 e8 74 46 db 18 0f ff 73`	(16 octets)

H:	`00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff`	(24 octets)
	`de ad be ef de af ba be`	
τ:	`17 99 c6 a8 df 70 85 17 23 04 cd 46 46 39 41 3c`	(16 octets)
M:	`00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f`	(52 octets)
	`10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f`	
	`20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f`	
	`fe fe ba be`	
C:	`2a 41 1f 68 c7 01 7c 54 85 2e 64 1c 81 02 ce a0`	(52 octets)
	`e3 59 bc e5 9f 76 59 0c 57 c9 c0 4a 98 14 63 5b`	
	`7d ef 80 62 5e ec 82 e7 66 17 4c 72 87 e7 d9 d4`	
	`a9 9b 6a 36`	
T:	`8b 83 74 f0 2b c6 de a1 98 a9 2a 8b 51 3b 60 42`	(16 octets)