Authenticated Encryption for Memory Constrained Devices

By
Megha Agrawal

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy
in Computer Science & Engineering

to the

Indraprastha Institute of Information Technology, Delhi (IIIT-Delhi)

Supervisors: Dr. Donghoon Chang (IIIT Delhi)
Dr. Somitra Sanadhya (IIT Jodhpur)

September 2020

Certificate

This is to certify that the thesis titled - **"Authenticated Encryption for Memory Constrained Devices"** being submitted by **Megha Agrawal** to Indraprastha Institute of Information Technology, Delhi, for the award of the degree of Doctor of Philosophy, is an original research work carried out by her under our supervision. In our opinion, the thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree/diploma.

Dr. Donghoon Chang

September, 2020
Department of Computer Science
Indraprastha Institute of Information Technology, Delhi
New Delhi, 110020

To my family

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisor Dr. Donghoon Chang for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Ph.D study.

I also express my sincere gratitude to my esteemed co-advisor, Dr. Somitra Sanadhya, who has helped me immensely throughout my Ph.D. life.

I thank my fellow labmates for the stimulating discussions, and for all the fun we have had in during these years.

I would like to thank my co-author Jinkeon Kang, for helping me in my work. I would also like to thank all my friends, especially Tarun for their help and cooperation that always kept my spirits high. Their company made my graduate life a memorable one.

I would also like to forward my gratitude to Tata Consultancy Services (TCS), India for awarding me the prestigious TCS fellowship for my full PhD period. Without their generous sponsorship, I would not have been able to travel around the world to attend a number of academic events, including conferences, workshops and summer schools.

I would also like to thank all the reviewers for providing valuable feedback to improve the quality of thesis.

Of course, none of this would be possible without my family who worked hard to sharpen my skills and have always encouraged and supported me to go further in life.

Above all, I am grateful to the Almighty, who showered the opportunity, blessings and moral courage on me to complete this dissertation.

Contents

List of Figures

List of Tables

Abstract

It is common knowledge that encryption is a useful tool for providing confidentiality. Authentication, however, is often overlooked. Authentication provides data integrity; it helps ensure that any tampering with or corruption of data is detected. It also provides assurance of message origin. Authenticated encryption (AE) algorithms provide both confidentiality and integrity / authenticity by processing plaintext and producing both ciphertext and a Message Authentication Code (MAC). It has been shown too many times throughout history that encryption without authentication is generally insecure. This has recently culminated in a push for new authenticated encryption algorithms. There are several authenticated encryption algorithms in existence already. However, these algorithms are sometimes difficult to use in resource constrained environment. This thesis focuses on designing of authenticated encryption schemes suitable for memory constrained environment.

In many practical applications, the users of an AE scheme use a cryptographic module to perform encryption, decryption and tag verification. Usually this cryptographic module has a very small memory. Due to its limited storage, it can't store the complete ciphertext to first verify the tag and then conditionally decrypt it. Similarly, it can't store the complete plaintext while decrypting, and output it only if the tag is valid. This becomes an issue which is particularly relevant in the case of long messages.

In authenticated encryption schemes, there are two techniques for handling long ciphertexts while working within the constraints of a low buffer size: Releasing unverified plaintext (RUP) or Producing intermediate tags (PIT). In this work, in addition to the two techniques, we propose another way to handle a long ciphertext with a low buffer size by storing and releasing only one (generally, or only few) intermediate state without releasing or storing any part of an unverified plaintext and without need of generating any intermediate tag. In this context we have designed two schemes sp-AELM which is sponge based and dAELM which is deterministic AE scheme. Brief details about these work are given below.

1. sp-AELM is a sponge based authenticated encryption scheme that provides support for memory constrined devices. We also provide its security proof for privacy and authenticity in an ideal permutation model, using a code based game playing framework. Furthermore, we also present two more variants of sp-AELM that serve the same purpose and are more efficient than sp-AELM.

2. We also analyzed sponge based CAESAR submissions using our proposed technique, to determine their potential to support limited memory constraint.

3. dAELM is a deterministic authenticated encryption scheme providing support for memory constrained device. Deterministic AE (DAE) is used in domains such as the key wrap, where the available message entropy omits the overhead of a nonce. For limiting memory usage, our idea is to use a session key to encrypt a message and share the session key with the user depending upon the verification of a tag. We provide the security proof of the proposed construction in the ideal cipher model.

4. We have shown a simple PA attack on an existing SAEB authenticated encryption scheme. Further, we have proposed a modification to SAEB to overcome this attack. We have proposed two modified versions of SAEB called RSAEB v1 and RSAEB v2 : first one provides PA1 security in nonce respecting scenario and another one provides PA1 in nonce misuse scenario and PA2. PA2 is stronger security notion than PA1, which comes at the cost of an additional pass.

Chapter 1

Introduction

The overarching goal of cryptography is to enable people to communicate privately over an insecure channel in the presence of adversaries. This notion maybe abstracted further; for example,the sending and receiving party may be the same person writing to and reading from a storage medium over time. Two requirements for achieving this goal are encryption and authentication. Encryption provides confidentiality while authentication provides data integrity and assurance of message origin. Encryption without authentication has been shown to be insecure in practical instances several times in recent history. For example, Wireless Equivalence Privacy (WEP), an algorithm for securing communications over wireless networks, is badly broken due (in part) to a lack of proper authentication [22].

In fact, it is so badly broken that people with no cryptographic expertise can use existing tools on standard consumer machines to break into networks that use 104-bit WEP in less than 60 seconds [76].

Another example of a flaw in prevalent systems due to a lack of proper authentication came in 2002. An attack was found on the CBC mode of encryption employed by protocols such as SSL and IPSEC. The attack could have been avoided had an authenticated encryption scheme been used [77]. Therefore, following a long tradition of cryptography competitions, CAESAR [2] aims to create a portfolio of authenticated encryption systems intended for wide public adoption.

1.1 Motivation

There exist a wide variety of *AE* modes. Some of the popular *AE* modes are GCM [58], OCB [67], EAX [13], CCM [33] and CWC [54]. In these "offline" *AE* modes, the device needs to store the complete plaintext, in the encryption mode, to produce the tag. Not all devices possess large memory to completely store a message and hence the modes mentioned earlier can't be directly used in the case of a long message. Consequently, the concept of online authenticated encryption was proposed in [38], in which encryption can be done on the fly. If the message (resp. ciphertext) blocks are denoted by $M_1, M_2, \ldots$ (resp. $C_1, C_2, \ldots$), then the requirement of online AE means that the ciphertext block C_i can be computed

without the knowledge of plaintext block M_j for any $j > i$. Most of the block-cipher based authenticated encryption schemes can be used in online encryption mode. Many authenticated encryption schemes that support online encryption have been submitted to the currently on going CAESAR competition [2]. Some of these are APE [4], NORX [50], AEGIS [78] etc.

However, decryption in an AE scheme can never be online due to the fact that the ciphertext needs to be verified before producing the plaintext. If this was not the case then an attacker could simply ask for the decryption of a ciphertext of his choice, while associating any arbitrary tag with the ciphertext. The requirement of verifying the tag before producing the decrypted text can be solved in two different ways. In one case, the system implementing the decryption process could store the complete plaintext and produce it only if the tag verifies. In the other case, the system first applies the tag verification algorithm and then produces the plaintext, block by block, only if the tag verifies. It may be possible to achieve the implementation of the first method in a single pass over the ciphertext, whereas the second method can't be implemented in less than 2 passes over the ciphertext. However, the first method requires potentially large memory on the device while the second method could be implemented even on low memory devices. In fact, with the increasing usage of low cost RFID devices [51], sensors and trusted platform modules (TPM) [53], the need for an AE scheme which can support both the encryption and the decryption functions in online form is increasing day by day. None of the schemes submitted in the CAESAR competition consider this limited memory constraint explicitly.

1.2 Outline of the thesis

In this thesis, we present new designs of authenticated encryption scheme suitable for memory constrained environment. We also analyze the security of some of the CAESAR submissions under the same environment. Our main results are presented in Chapters 7, 8 and 9. Each chapter provides literature review, followed by results and conclusions. The thesis outline is as follows:

- In Chapter 2, we begin by giving basic definitions of the main security goals and elaborate on the major two: confidentiality and authenticity. In Section 2.2, we explain the standard communication model, in which the fulfillment of the goals is demanded.

- In Chapter 3, we explain the the basics of authenticated encryption and their construction approaches. We then look at the security notions for authenticated encryption.

- In Chapter 4, we define the basic preliminaries required to understand security proof. We explain the basics of provable security, adversarial models and framework used to write proof.

- In Chapter 5, we define the motivation behind our work. We describe the need of cryptographic module to implement the authenticated encryption and associated issues with it. We also explain the operational environment which we will consider further in the chapters to design an authenticated encryption scheme.
- After motivating the problem in previous chapter, in next Chapter 6, we give brief overview of the existing works in this direction, followed by their limitations.
- In Chapter 7, we proposed a new sponge based AE scheme sp-AELM to support cryptographic modules having limited storage capabilities. We provided its security proof in an ideal permutation model using code based game playing framework for both privacy and authenticity. In addition to this, we also present two more variants of sp-AELM that serve the same purpose and are more efficient than sp-AELM.
- In Chapter 8, we applied this newly introduced technique to all Sponge based AE schemes submitted to the CAESAR competition for determining their suitability to support devices with memory constraint.
- In Chapter 9, we present a new deterministic authenticated encryption scheme dAELM which is suitable for the memory constrained scenarios.
- In Chapter 10, we present a authenticated encryption scheme RSAEB, which is secure in RUP (Release Unverified Plaintext) settings.
- Finally in Chapter 11, we conclude the thesis by discussing the results and giving some future directions of research.

Chapter 2

Basics

In today's computer-centric world, information technology is widely used for storing and transmission of data over different channels. Most of the times, this transmission of data is over insecure networks such as internet, or via satellites. Some examples of application where sensitive information is sent through these insecure channels include web browsing, payment gateways, smart cards to protect information from financial transactions, RFID transponders, devices in sensor networks and electronic identity card. To protect this information, we require security mechanisms. Cryptography is the science that is used to protect the sensitive information over insecure channels.

2.1 Cryptographic Goals

Cryptography aims at achieving information security and for this purpose, we define various security goals such as confidentiality, data integrity and availability. These are known as major cryptographic goals and are formally defined as follows.

1. ***Confidentiality***: ensuring that an adversary listening to the communication channel can not gain information about the content of information.
2. ***Data integrity***: ensuring that any adversary with access to communication channel is unable to manipulate the contents of some communication by unauthorized means.
3. ***Data origin authentication***: ensuring that an adversary with access to communication channel is unable to modify and/or misinterpret the true origin of some communication. Property of data origin authentication directly implies data integrity since the message that has been modified has a new source.
4. ***Non-repudiation***: ensuring that someone can not deny the validity of previous commitments or actions.
5. ***Availability***: ensuring that data is available to the authorized user in the required manner without any delay.

These security goals can be achieved using different cryptographic primitives. For example, confidentiality can be achieved using encryption algorithms, data integrity can be achieved using hash functions. Our research is pointed towards the study of those cryptographic primitives which achieve confidentiality and authenticity.

2.2 Communication Model

One aspect of information security is communication security. In this section we present the standard model of communication between parties and in next section we will present how this communication model can be violated with respect to different security goals. We used a well established transmission model given by Shannon and Weaver [71] as a standard model of communication.

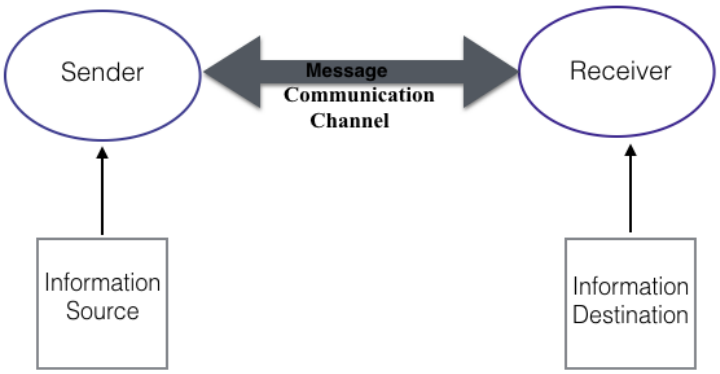


Figure 2.1: Standard Communication Model

The model consists of five parts:

1. An *information source* which produces a message or sequence of messages to be communicated to the receiver.
2. The *transmitter* or sender which operates on a message in some way to make it suitable for transmission over a channel.
3. The *channel* is the medium used to transmit the signal from sender to receiver.
4. The *receiver* performs the inverse operation of that done by the sender to reconstruct a original message.
5. The *destination* is the person for whom message is intended.

2.2.1 Generic Attacks on Communication Model

In above defined communication model, any unauthorized third party with malicious intentions is known as an attacker or adversary. Their goal is to get some useful information over a communication channel while two parties are interacting.

There are three types of generic attack that can be carried out by an adversary to attack on the basic cryptographic goals: confidentiality, authenticity, availability.

1. Attack on confidentiality: An adversary just monitors all the information travelling over a communication channel between two parties as shown in Fig. 2.2.

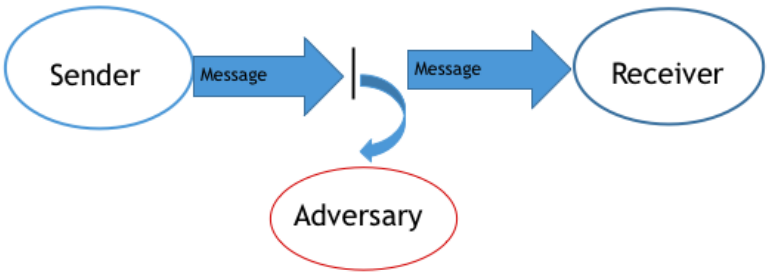


Figure 2.2: Attack on Confidentiality

2. Attack on authenticity: In impersonation attack, an adversary pretends to be valid sender of a new message. A substitution attack is when the adversary modifies the usual communication, so that receiver does not get the intended message from the sender, but a modified one. This attack is shown in Fig. 2.3.

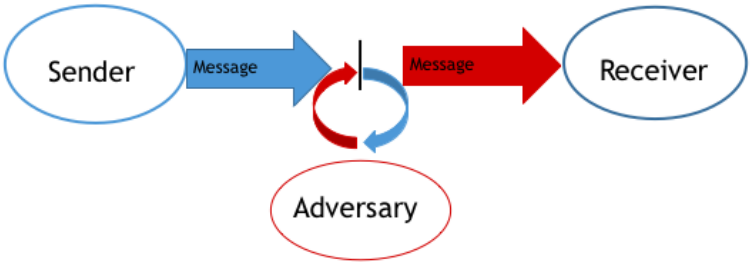


Figure 2.3: Attack on Authenticity

3. Attack on availability: An adversary stops the usual communication, so that message does not reach the receiver. This is shown in Fig. 2.4.

Figure 2.4: Attack on Availability

Based on the role of an adversary in above attacks, these attacks can be classified in two categories:

1. ***Active attack***: is an attack where adversary attempts to add, delete or modified the transmitted data. An active attack threatens the confidentiality, integrity and authenticity.

2. ***Passive attack***: is an attack where the adversary only monitors the communication channel. It only threatens the confidentiality of the data.

2.3 Basic settings of Cryptography

Cryptography enables to store sensitive information or transmit it across insecure networks (like the Internet) so that it cannot be read by anyone except the intended recipient. Basic settings of cryptography consist of two parties communicating over a insecure channel. These two party includes a sender, usually known as "Alice" and a receiver known as "Bob". Alice encrypt the data using a Key and produce ciphertext(known as "Encryption") and send this ciphertext over the network. On the other side, Bob received ciphertext and decrypt it using the key and produce plaintext(known as "Decryption").

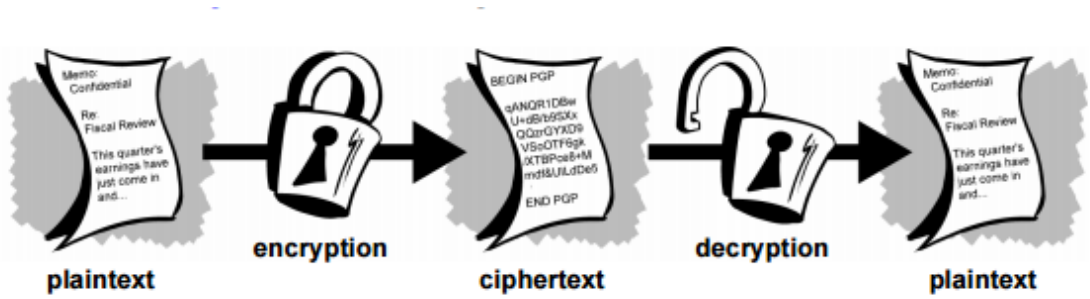


Figure 2.5: Encryption and Decryption

Depending on the number of keys that are employed for encryption and decryption, cryptography can be divided into 2 categories:

1. Private key cryptography: In private key cryptography, also called secret-key or symmetric key cryptography, one key is used both for encryption and decryption. This key is preshared by both parties over secure network.

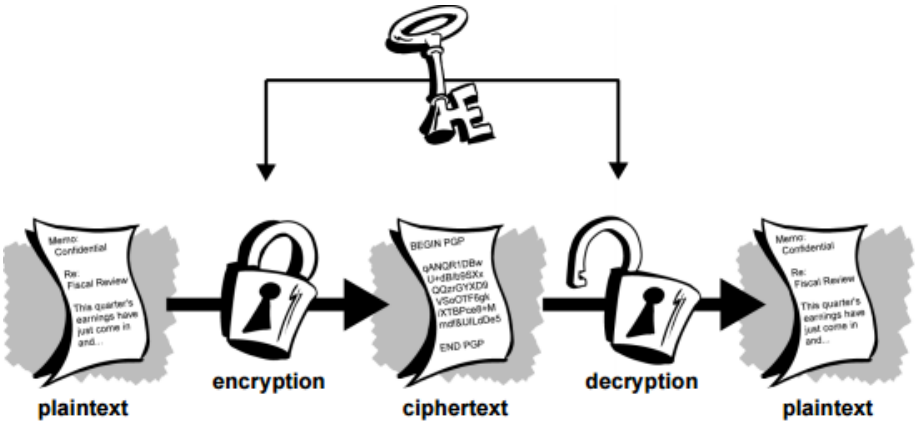


Figure 2.6: Symmetric key cryptography settings

2. Public key cryptography: Public key cryptography also known as asymmetric cryptography uses a pair of keys : a public key, which encrypts data, and a corresponding private, or secret key for decryption.

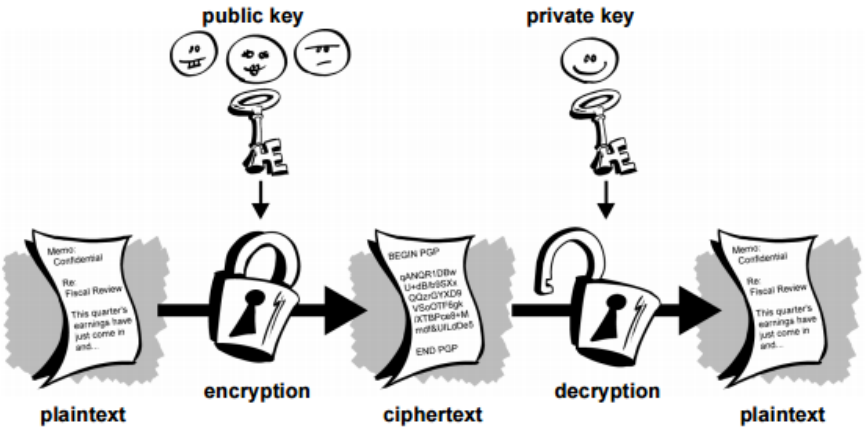


Figure 2.7: Public key cryptography settings

In this report, our focus will be on symmetric encryption only. We will discuss everything in context of symmetric key cryptosystem.

2.4 Symmetric Key Cryptosystem

As we discussed above, in symmetric key settings, both communicating parties use the same key for communication and we assume that this key is shared between both parties over secure channel and not any third party has access to it.

In the symmetric key cryptography settings, there are two important security properties which need to be satisfied for a communication over insecure channel. These two security properties include confidentiality (also know as privacy) and Authenticity. We will discuss both of these properties briefly here.

2.4.1 Confidentiality

The goal of a symmetric encryption scheme is that an adversary who obtains the ciphertext C, must be unable to learn anything about the plaintext M(except the length of plaintext). In other words, the ciphertext looks like a random string for those who does not know key K, even if they have some control over the plaintext.

Confidentiality can be easily achieved in a system by using any encryption algorithm. Here we define a general symmetric key encryption algorithm. A typical symmetric encryption scheme Π is defined as tuple of three algorithms $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The first algorithm $\mathcal{K}$ is the key generation algorithm which specifies the manner in which the key K is to be chosen. In most cases this algorithm simply returns a random string of key length. Second algorithm $\mathcal{E}$ is a encryption algorithm, which takes user supplied plaintext M as an input and generate ciphertext C as output using key K. The encryption algorithm $\mathcal{E}$ may be randomized, or it might keep some state around. And the last algorithm in Π is decryption algorithm $\mathcal{D}$, which take ciphertext C as input and decrypt it using key K. The decryption process might be unsuccessful, indicated by its returning a special symbol $\perp$, but, if successful, it ought to return the message that was originally encrypted. And the decryption algorithm $\mathcal{D}$ is always deterministic.

From computational point of view, the security of a scheme is defined in terms of the probability to break some security notion relating to the scheme. As mentioned before, only the key is kept secret from the adversary. If the encryption scheme has a key length of n bits, then with probability 2^{-n}, the adversary can find the correct key in a single attempt. Once the key is known, then the adversary can decrypt the ciphertext as if he was the legitimate recipient.

2.4.2 Authenticity

In the message-authentication problem the receiver gets some message M which is claimed to have originated with a particular sender. The channel on which this message flows is insecure. Thus the receiver of the message M wants to distinguish the case in which the message really did originate with the claimed sender from the case in which the message originated with some imposter. In such a case we consider the design of

an encryption algorithm with the property that un-authentic transmissions lead to the decryption algorithm outputting the special symbol $\perp$.

The most common tool for solving the message-authentication problem in the symmetric setting is a message authentication scheme, also called a message authentication code (MAC). Such a scheme is specified by a triple of algorithms, $\Pi = (\mathcal{K}, \mathcal{T}, \mathcal{V})$. When the sender wants to send a message M to the receiver she computes a "tag, σ, by applying $\mathcal{T}$ to the shared key K and the message M, and then transmits the pair (M, σ). (The encryption procedure referred to above thus consists of taking M and returning this pair. The tag is also called a MAC.) The computation of the MAC might be probabilistic or use state, just as with encryption or it may be deterministic. The receiver, on receipt of M and σ, uses the key K to check if the tag is valid by applying the verification algorithm $\mathcal{V}$ to K, M and σ. If this algorithms returns 1, he accepts M as authentic; otherwise, he regards M as a forgery.

2.5 Providing Confidentiality

Encryption schemes are cryptographic primitives that are designed to achieve the goal of confidentiality as mentioned in above section. Depending on the length of input message M, symmetric encryption schemes are classified as block ciphers and steam ciphers.

2.5.1 Block Ciphers

As the name implies, block ciphers operate on *blocks* of data. These blocks are typically 64 or 128 bits in length and they are transformed into blocks of the same size under the action of a secret key. The block cipher encrypts a block of plaintext or message m into a block of ciphertext c under the action of a secret key k, which is denoted as $c = Enc_k(m)$. The process of encryption is reversed by decryption, which will use the same user supplied key. This is denoted as $m = Dec_k(c)$.

Block ciphers are public and fully specified algorithms. One of the most popular block cipher for the past 20 years has been the Data Encryption Standard (DES) [26] developed in 1973 by IBM. DES was subsequently widely adopted and has been published in many standards.Although DES has a good design, its key is quite short, 56-bits, which makes it vulnerable to exhaustive key search attack (2^{56} possible keys) as shown by W. Diffie and M. Hellman in [31, 43]. DES has also been theoretically broken using a modern cryptanalysis technique called Differential Cryptanalysis. Later, Rijandel's block cipher, better known as Advanced Encryption Standard (AES) [28] has generated considerable interest. This is a block cipher which can work with 128, 256 and 512 bit keys, and has 10-14 rounds of simple block ciphers. The AES is the current encryption standard by FIPS-197 [27], intended to be used by U.S. government organizations to protect sensitive (and even secret and top secret) information. It is also becoming a global standard for commercial software and hardware that use encryption or other security features.

Block ciphers process message blocks of same length. Plaintext messages exceeding one

block in length, are processed by modes of operation, which are cryptographic primitives based on block ciphers.

2.5.2 Stream Ciphers

Stream ciphers encrypt bits individually. This is achieved by adding a bit from a key stream to a plaintext bit. There are synchronous stream ciphers where the key stream depends only on the key, and asynchronous ones where the key stream also depends on the ciphertext. If the dotted line in Fig. 2.8 is present, the stream cipher is an asynchronous one.

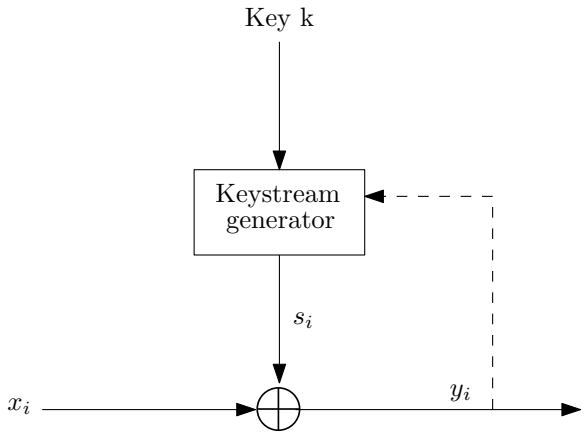


Figure 2.8: Synchronous and Asynchronous Stream Cipher

The best known general purpose stream cipher is RC4 [70]. It has a variable key length with a short, 40-bit key, or with a longer 128-bit key.

2.6 Providing Authenticity

Data integrity is strongly related to data origin authentication. If a message is altered, the integrity has been attacked (and compromised). However, it is also clear that its originator is no more the authenticated sender and vice versa. This implies that message integrity and origin authentication are inseparably bound together. A violation of one automatically leads to doubtfulness for the other. Hence, authenticity mechanisms implicitly provide data origin authentication and integrity.

2.6.1 Hash Function

Hash functions map an arbitrarily long string into a fixed number of output bits, called hash value. Therefore, it is important for the hash function to be collision-free, not only should it be nearly impossible to retrieve the original data. It must also be unfeasible to construct a data block that matches some given hash value. Hash functions should be completely

deterministic i.e. given the same input twice, the hash function should always produce the same output.

The basic idea behind hash functions is that the hash value serves as a representative image of the input string and that it is uniquely identifiable with that string, this way warranting the integrity of the string.

2.6.2 Message Authentication Code (MAC)

MACs are a special and primary class of message authentication schemes and a particular application of keyed hash functions. The usage of authentic channels here can be avoided, because MACs have a secret parameter, the key. A MAC is defined as a set of algorithms $MAC = (Gen, Tag, Ver)$.

1. The *randomized key generation algorithm* Gen returns a string K, $K \leftarrow Gen$, where $K \in Keys(MAC)$. $Keys(MAC) = \{0,1\}^k$ is the set of all strings that have non-zero probability of being output by the algorithm Gen. Keys are generated for each party involved in the communication.

2. The *tagging algorithm* Tag produces a tag $T = Tag_K(M)$ for any message $M \in \mathcal{M}$. The tagging algorithm is the actual application of the keyed hash functions, or $Tag : Keys(MAC) \times \mathcal{M} \rightarrow \mathcal{T}$. A tag function takes the random key K generated by Gen and a message $M \in \mathcal{M}$, where $\mathcal{M} = \{0,1\}^*$, and computes $Tag_K(M) = T$, such that $T \in \mathcal{T}$ and $\mathcal{T} = \{0,1\}^\tau$ is the set of binary strings of a fixed length τ, where $\tau \geq 1$.

3. The *Verification algorithm* Ver verifies whether the integrity of the message is violated, by comparing the sent tag T with the tag T' calculated by the receiver. Both sender and receiver use same tagging algorithm for computing their tags. The algorithm produces a value $Ver_K(M,T) \in \{\perp, \top\}$, which outputs $\top$ if T is a valid tag, otherwise it outputs $\perp$.

The authenticity of a message M is confirmed when the receiver of the pair (M,T) verifies the tag T for the given message M.

Chapter 3

Authenticated Encryption

3.1 Introduction

In the last chapter, we studied about individual primitives for achieving confidentiality and authenticity. However, in practice we often want both privacy and authentication simultaneously. Authenticated encryption (AE), formalized in in [9, 11], is a cryptographic primitive with pre-shared key providing both data privacy and authenticity at the same time.

In practice, there could be scenario, where we need some additional information like packet header that travel alongside with the ciphertext and need authentication only, this type of information is called associated data. And the AE scheme that supports the use of associated data is called authenticated encryption with associated data (AEAD).

The design of Authenticated encryption (AE) scheme gained enormous interest among the cryptographic research community, after the announcement of the CAESAR [2] competition. CAESAR (Competition for Authenticated Encryption: Security, Applicability, and Robustness) was an effort to identify a portfolio of authenticated ciphers that are suitable for widespread adoption and offer an advantage over AES used in Galois/counter mode [58].

In this chapter we first focus on the motivation of AE usage by discussing their necessity and applications. Then, we move towards descriptions of the particular designs and functionality of some of the well known AE modes.

3.2 Necessity and Usage

Confidentiality and authentication are vital features that must be present in any communications network. The need for these is increased in the case of wireless communications than in wired communications, because of the shared nature of the wireless medium. The nature of the wireless medium is such that it is more susceptible to adversarial attacks. In the wireless medium, anyone can listen to whatever is being sent over the network. Also, the presence of a shared communication medium does not uniquely identify

the originator. To make things worse, any tapping or eaves-dropping cannot always be detected in the wireless medium. Thus, confidentiality coupled with authentication plays an important role especially for the successful operation of the wireless communication systems, e.g. Internet and mobile systems.

Integrated mechanisms that ensure the latter security requirements are the AE schemes. In particular, AE schemes find important usage in Internet protocols and applications like:

1. **Secure Shell (SSH) protocol** [80]: SSH is an IETF draft protocol used for securing terminal sessions and authentications over an insecure net- work at the application level. The cryptographic heart of the SSH protocol is its binary packet protocol (BPP) [80], which is responsible for the underlying symmetric encryption and authentication (or authenticated encryption) of all messages sent between two parties involved in an SSH connection. Before beginning the authenticated encryption portion of an SSH session, a client and a server first agree upon a set of shared symmetric keys (a different set for each direction of a connection). The client and the server also agree upon which encryption and message authentication schemes they wish to use. The encryption schemes recommended by Ylonen in [79] are based on CBC encryption mode, and the recommended message authentication scheme is HMAC [8].

2. **Secure Sockets Layer (SSL)** [18]: The SSL protocol was developed by Netscape Communications Corporation to provide privacy and authentication over the Internet. The protocol also supports server and client authentication. The SSL protocol is application independent, allowing protocols like HyperText Transfer Protocol (HTTP) [37], File Transfer Protocol (FTP) [63], and Telnet [64] to be layered on top of it transparently. The SSL protocol is able to negotiate encryption keys as well as authenticate the server before data is exchanged by the higher-level application. The SSL protocol maintains the security and integrity of the transmission channel by using encryption and authentication mechanisms.

3. **Transport Layer Security (TLS)** [30]: The TLS protocol also uses data encryption and authentication mechanisms to ensure privacy between communicating applications and authentication for their users on the Internet. When a server and client communicate, TLS ensures that no third party may eavesdrop or tamper with any message. TLS is the successor to SSL. TLS is composed of two layers: the TLS Record Protocol and the TLS Handshake Protocol. The TLS Record Protocol provides connection security with some encryption method. The TLS Record Protocol can also be used without encryption. The TLS Handshake Protocol allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys be- fore data is exchanged. Both SSL and TLS support MD5 [65] and SHA-1 [34] hash functions for data integrity and the CBC encryption mode and RC4 stream cipher for the encryption part. But while SSL uses simply the hash functions for data integrity, TLS in addition uses HMAC to provide a better symmetric authentication strength.

4. **IP Security Protocol (IPSec)** [46]: The IETF's IP Security Protocol working group defines a set of specifications for cryptographically based authentication, integrity, and confidentiality services at the IP datagram layer. IPSec is intended to be the future standard for se- cure communications on the Internet, but is already the de facto standard. The IPSec group's results comprise of a basis for interoperably secured host-to-host pipes, encapsulated tunnels, and Virtual Private Networks (VPNs), thus providing protection for client protocols residing above the IP layer. The IPSec protocol relies also on the HMAC- SHA and HMAC-MD5 for ensuring data integrity and CBC encryption mode with different underlying block ciphers (DES, Triple DES, AES, Twofish, etc.) for the encryption part.

5. **Online applications**: These are mostly commercially and personally secure applications. The commercially secure applications involve transactions (conversations), where proprietary information is discussed, while the personally secure ones include personal conversations or data that is not necessarily confidential.

   (A) Examples for commercially secure applications:
      (a) Online banking and online retail.
      (b) Online auctions, shopping, acquisitions, contact negotiations, stock transactions, mergers, etc.

   (A) Examples for personally secure applications:
      (a) Instant messaging, video and audio online communication, remote login, secure file and message transfer, etc.

From the brief summary above, we observe that the aim of ensuring both confudentiality and authenticity is an obvious need. While this was underesti- mated in the past, when more emphasis was put on the privacy mechanisms, nowadays both security conditions are absolutely required to present and achieve effectively their goals wherever needed.

3.3 Definition

Authenticated Encryption has been informally introduced in the previous section. Now we formally define AE.

Definition 1. ***(AE scheme)*** *An AE scheme is defined as* $\Pi = (\mathcal{K}, \mathcal{E}nc, \mathcal{D}ec)$. $\mathcal{K}$ *is the key generation algorithm. Usually a random string,* K *of key length is produced as its output.* $\mathcal{E}nc$ *is called encryption algorithm which takes* K*, a nonce* N*, message* M *as input and outputs a* (C, T) *ciphertext and tag pair. Decryption algorithm,* $\mathcal{D}ec$ *using* K*, nonce* N*, ciphertext and tag pair* (C, T)*, decrypts the ciphertext to plaintext and verifies whether the given tag matches. If it matches then it outputs the plaintext* M *otherwise it outputs a special symbol (say INVALID or* $\perp$*) to signify that the authentication failed. This can de given as:*

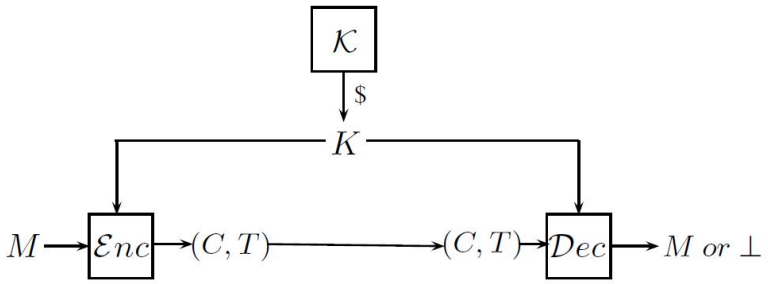


Figure 3.1: Authenticated Encryption

$$\begin{aligned} \mathcal{E}nc &: \mathtt{K} \times \mathtt{N} \times \mathtt{M} \to (\mathtt{C}, \mathtt{T}) \\ \mathcal{D}ec &: \mathtt{K} \times \mathtt{N} \times \mathtt{C} \times \mathtt{T} \to \mathtt{M}/\perp \end{aligned}$$

with $\mathtt{K}$ the keys, $\mathtt{N}$ the nonces, $\mathtt{C}$ the ciphertexts, $\mathtt{T}$ the tag and $\perp$ an invalid symbol which represents verification failure. The correctness condition states that for all $K \in \mathtt{K}$, $N \in \mathtt{N}$ and $M \in \mathtt{M}$,

$$\mathcal{D}ec_K^N(\mathcal{E}nc_K^N(M)) = M$$

Pictorial view of AE is shown in Fig. 3.1. In this figure, we haven't considered the nonce N.

Extend to AEAD schemes: Authenticated encryption schemes with associated data, AEAD, are an extended version of AE schemes. Some of the AE schemes that have the additional property of processing static data (that usually contains IP Address of sender, receiver, number of packets sent, etc.) called header or associated data A. This data often does not need to be encrypted, but requires authentication. Such schemes that provide only authentication for the header part and authenticated encryption for the rest of the message, are called AE schemes with associated data.

Definition 2. ***(AEAD scheme)*** *An AEAD scheme is defined as* $\Pi = (\mathcal{K}, \mathcal{E}nc, \mathcal{D}ec)$. *$\mathcal{K}$ is the key generation algorithm. Usually a random string, K of key length is produced as its output. $\mathcal{E}nc$ is called encryption algorithm which takes K, a nonce$N$, a associated data A and outputs a (C, T) ciphertext and tag pair. Decryption algorithm, $\mathcal{D}ec$ using K, nonce N, a associated data A, ciphertext and tag pair (C, T), decrypts the ciphertext to plaintext and verifies whether the given tag matches. If it matches then it outputs the plaintext M otherwise it outputs a special symbol (say INVALID or $\perp$) to signify that the authentication failed. This can de given as:*

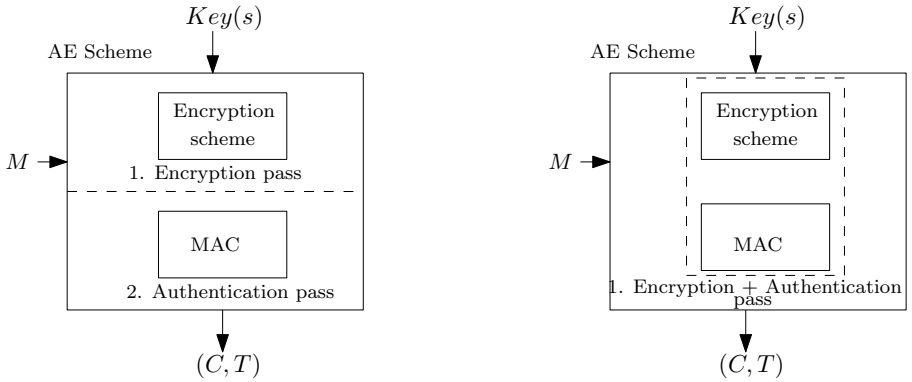
$$\begin{aligned} \mathcal{E}nc &: \mathtt{K} \times \mathtt{N} \times \mathtt{A} \times \mathtt{M} \to (\mathtt{C}, \mathtt{T}) \\ \mathcal{D}ec &: \mathtt{K} \times \mathtt{N} \times \mathtt{A} \times \mathtt{C} \times \mathtt{T} \to \mathtt{M}/\perp \end{aligned}$$

with K the keys, N the nonces, A the associated data, C the ciphertexts, T the tag and $\perp$ an invalid symbol which represents verification failure. The correctness condition states that for all $K \in \mathtt{K}$, $N \in \mathtt{N}$, $A \in \mathtt{A}$ and $M \in \mathtt{M}$,

$$\mathcal{D}ec_K^{N,A}(\mathcal{E}nc_K^{N,A}(M)) = M$$

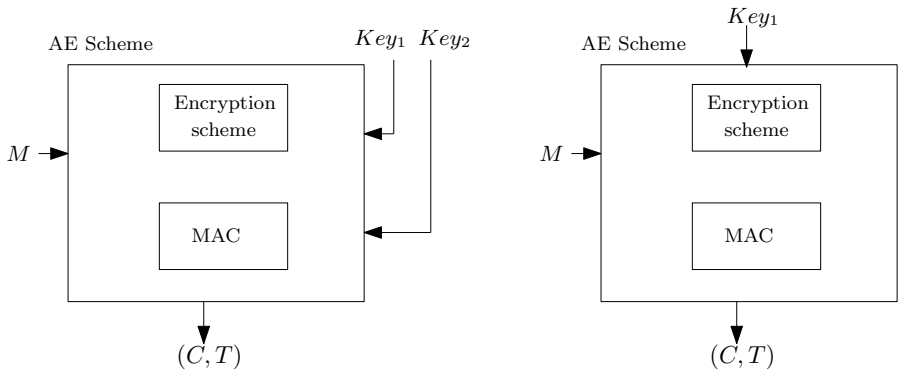
3.4 AE Classification

An AE schemes can be classified based on the certain characteristics. We have given the AE classification that distinguishes the scheme by following features:



(a) Conventional AE Scheme

(b) Dedicated AE Scheme



(c) Generic AE Scheme

(d) Non-generic AE Scheme

Figure 3.2: Conventional, Dedicated, Generic, Non-generic AE schemes in black-box design

- **Number of passes over data**: The number of passes specifies the way encryption and authentication primitives are applied on the data. In case of two primitives where encryption and authentication are applied individually on data (in two passes), then

the AE schemes are referred to as conventional 3.2(a). On the other hand, AE designs, where the encryption and authentication of data is carried out simultaneously, in one go over the data, are known as dedicated designs 3.2(b).

- **Way of achieving encryption and authentication**: Under this category, AE scheme can be classified in 2 ways: generic 3.2(c) and non-generic 3.2(d). The generic designs combine an autonomous encryption scheme with an authentication one in a given order. Most characteristic for those designs is the usage of two separate keys for each scheme. The non-generic designs can be built as an extension of the generic one either by adding specific properties and functionalities to the combined AE design, such that it becomes unique, or they can simply be new mechanisms, specially introduced to achieve authenticated encryption. The non-generic designs use a single key.

In Fig.3.3, we showed the above described classification of the AE schemes. In this chart, we have only included the well known AE schemes.

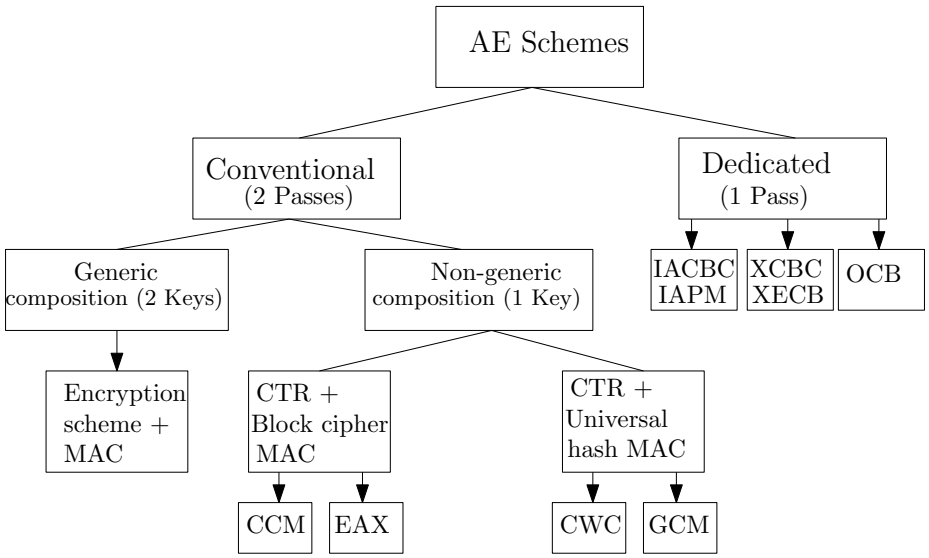


Figure 3.3: Classification of the basic AE modes of operation

3.5 Conventional AE Schemes

The simplest approach to use encryption and authentication schemes in a conventional way is the generic composition approach.

3.5.1 Generic Compositions

There are 3 ways to combine secure any encryption and authentication(MAC) scheme. We will discuss all of three one by one.

Let $\mathcal{E} = (E, D)$ be a cipher defined over $(\mathcal{K}, M, C)$ and let $\mathcal{I} = (S, V)$ be a MAC defined over $(\mathcal{K}, C, T)$. To keep notation simple, we assume both E and $\mathcal{I}$ uses the key in same key space $\mathcal{K}$.

1. **Encrypt-and-Authenticate (E&A)**

   The Encrypt-and-MAC system or $E\&M$ in short is defined as follows: $\Pi_{E\&M} = (\mathcal{K}_{E\&M}, \mathcal{E}_{E\&M}, \mathcal{D}_{E\&M})$ where $\mathcal{K}_{E\&M}$ generates two independent random keys k_e and k_m and $\mathcal{E}_{E\&M}$, $\mathcal{D}_{E\&M}$ are defined as follows:

   $$\begin{aligned} \mathcal{E}_{E\&M}((k_e, k_m), M) &:= C \leftarrow E(k_e, M), \quad T \leftarrow S(k_m, M) \\ &\quad \text{output } (C, T) \\ \mathcal{D}_{E\&M}((k_e, k_m), (C, T)) &:= M \leftarrow D(k_e, C) \\ &\quad \text{if } V(k_m, M, T) = \texttt{reject} \text{ then output } \texttt{reject} \\ &\quad \text{otherwise, output } M \end{aligned}$$

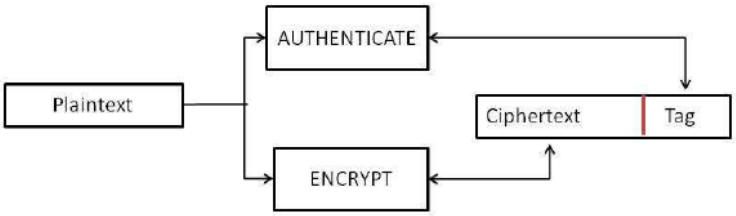
   

   Figure 3.4: Encrypt-and-Authenticate

   This mode of AE is not considered to be secure. An example showing the insecurity of this scheme is given in [9]. The transport layer of SSH uses a variant of this method.

2. **Encrypt-then-Authenticate (EtM)**

   The Encrypt-then-MAC system or EtM in short is defined as follows: $\Pi_{EtM} = (\mathcal{K}_{EtM}, \mathcal{E}_{EtM}, \mathcal{D}_{EtM})$ where $\mathcal{K}_{EtM}$ generates two independent random keys k_e and k_m and $\mathcal{E}_{EtM}$, $\mathcal{D}_{EtM}$ are defined as follows:

   $$\begin{aligned} \mathcal{E}_{EtM}((k_e, k_m), M) &:= C \leftarrow E(k_e, M), \quad T \leftarrow S(k_m, C) \\ &\quad \text{output } (C, T) \\ \mathcal{D}_{EtM}((k_e, k_m), (C, T)) &:= \text{if } V(k_m, C, T) = \texttt{reject} \text{ then output } \texttt{reject} \\ &\quad \text{otherwise, output } D(k_e, C) \end{aligned}$$

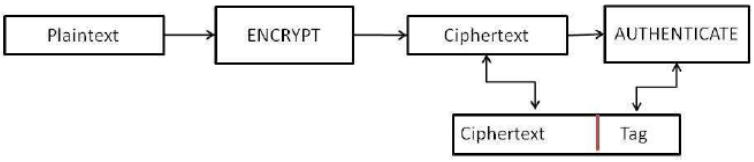


Figure 3.5: Encrypt-then-Authenticate

This mode of AE is proven to be secure in [9]. IPSEC uses a variant of this method.

3. **Authenticate-then-Encrypt (AtE)**

   The Authenticate-then-encrypt system or MtE in short is defined as follows: $\Pi_{MtE} = (\mathcal{K}_{MtE}, \mathcal{E}_{MtE}, \mathcal{D}_{MtE})$ where $\mathcal{K}_{MtE}$ generates two independent random keys k_e and k_m and $\mathcal{E}_{MtE}$, $\mathcal{D}_{MtE}$ are defined as follows:

$$\begin{aligned} \mathcal{E}_{MtE}((k_e, k_m), M) &:= T \leftarrow S(k_m, M), \quad C \leftarrow E(k_e, (M, T)) \\ &\quad \text{output } C \\ \mathcal{D}_{MtE}((k_e, k_m), (C, T)) &:= (M, T) \leftarrow D(k_e, C) \\ &\quad \text{if } V(k_m, M, T) = \texttt{reject} \text{ then output } \texttt{reject} \\ &\quad \text{otherwise, output } M \end{aligned}$$

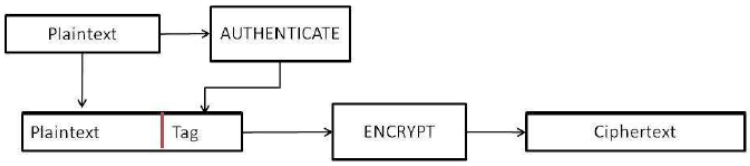


Figure 3.6: Authenticate-then-Encrypt

This mode of AE is also not secure. An example showing the weakness of this mode is available in [9]. SSL uses a variant of this method.

3.5.1.1 Properties of the Generic Compositions

1. **General**

   - usage of two keys: a considered disadvantage of a AE scheme, which requires more memory storage space, additional key man- agement and scheduling;
   - managing the functionality of two separate primitives: the com- plex functionality of an AE scheme would be dependable on the particular properties of the underlying primitives;

2. **Efficiency**

   - block cipher runs: two independent runs, one for encryption and one for authentication. Adds up increased costs;
   - verification: we distinguished the order of the decryption and verification followed by the receiver, which is useful information for establishing the rejection speed of forged messages and the average cost of decryption. Fast verification is achieved in the case of first verifying and then decrypting the message. This order of processing in decryption allows immediate check for malicious interventions and therefore saves the additional cost of unnecessary decryption in the given scenario. The generic construction of Encrypt Then MAC is hence most efficient in this respect.

3. **Simplicity**: The generic compositions have relatively simple design as they simply combine two well known primitives.

4. **Security results**: These are based on the security assumptions for the underlying primitives. The security results for the three generic compositions are deeply analyzed by Bellare et al. in [9] and here we give a brief summary in the following Table 3.1((secure is denoted by + , insecure as - and WMAC stays for weakly and SMAC for strongly unforgeable MAC)).

Composition		E&A		AtE		EtM	
Security		WMAC	SMAC	WMAC	SMAC	WMAC	SMAC
Privacy	IND-CPA	-	-	+	+	+	+
	IND-CCA	-	-	-	-	-	+
	NM-CPA	-	-	-	-	-	+
Authenticity	INT-PTXT	+	+	+	+	+	+
	INT-CTXT	-	-	-	-	-	+

Table 3.1: Security results for the generic composition based AE schemes as per [9].

The strongest security is achieved by the Encrypt then MAC scheme. This composition method provides security, under the assumption that the underlying encryption scheme achieves IND-CPA and the MAC is strongly unforgeable. In other words, having a secure encryption scheme together with a secure MAC, the Encrypt then MAC composition technique implies security of the generated scheme and no further security analysis is needed. In case other composition technique is used, like MAC then En- crypt or Encrypt and MAC, the combined AE scheme is not immediately considered to be secure. A security check for all possible instantiations of the underlying constituent primitives is needed, for an AE scheme to be proven secure.

3.5.2 Non-generic Compositions

Next we present the non-generic conventional AE schemes. All of them are nonce-based AE designs, using the CTR encryption mode of operation together with either a block cipher based MAC (CCM and EAX designs) or an universal hash function based MAC (CWC and GCM designs).

3.5.2.1 Counter mode with CBC-MAC: CCM

D. Whiting, R. Housley, and N. Ferguson present CCM in [33]. CCM is an AEAD mode that is a mandatory encryption algorithm in the IEEE 802.11 draft standard. CCM uses CTR mode for encryption and CBC-MAC for message authentication with AES underlying block cipher.

Only one underlying key K is used for both the authentication and the encryption part. An additional plaintext data A can be authenticated. A variable length nonce N, unique for each message, is combined with flags and counter ad serves as an input for a key stream generation in CTR mode. The message integrity is verified using the authentication tag T with variable length of 4, 6, 8, 10, 12, 14 or 16 bytes. CCM mode is approved specially for the ciphers with 128 bits block length. An encryption under CCM mode is shown in Fig. 3.7. For a detailed specification one can refer to [33].

The advantage of this mode is that it combines two well known cryptographic primitives, i.e. CTR mode encryption and CBC-MAC authentication. Moreover, CCM uses a single key for all block cipher operations. Still, the mode adds excessive bit manipulations and the simple primitives remain hidden behind complex bit transformations. The complexity of the mode raises certain criticism against its usage and attempts for standardization. A reasonable analysis on the disadvantages of the mode is done by P. Rogaway and D. Wagner in [69]. They outline the basic weaknesses of the CCM with respect to efficiency, complexity, choice of parameters, and security claims.

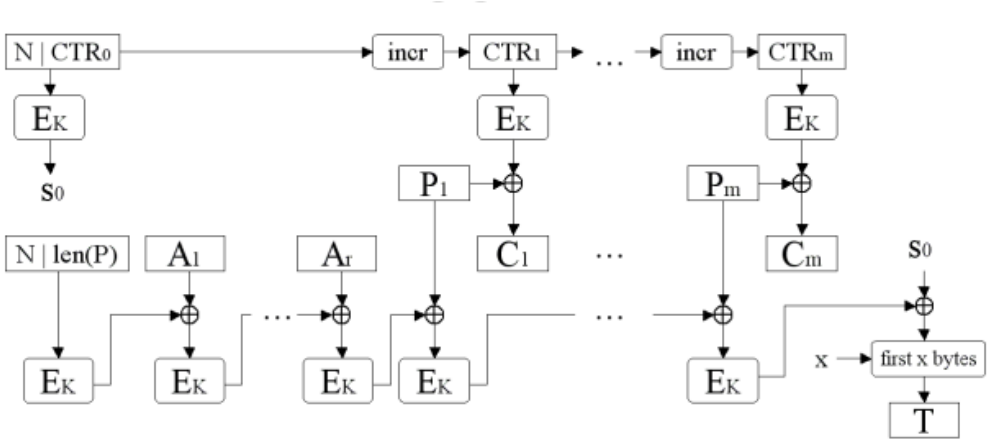


Figure 3.7: CCM Mode of operation [73]. The plaintext is $P = P_1 \dots P_m$, the key is K and the associated data or header information is $A = A_1 \dots A_r$. The ciphertext is $C||T$ where $C = C_1 \dots C_m$.

CCM Properties

1. **General**

   - Single key usage.
   - It lacks the ability to preprocess static headers or associate data as it encodes the nonce and the message length before authenticating the header.
   - Message length: multiple of 8 bits.
   - Non-repeating nonce usage of varying length from 56 to 104 bits.

2. **Efficiency**

   - CCM requires the length of the message to be known in advance, which makes the incapable of online processing.
   - Fast verification: CCM first verifies and then decrypts the data.
   - Big number of block cipher calls : $2\lceil \frac{|P|}{128} \rceil + \lceil \frac{|A|}{128} \rceil + 2 + \delta$, for $\delta \in \{0, 1\}$ (the given formula is valid for static headers as well). It is a consequence of the usage of block cipher based CBC-MAC, which requires minimum twice as many block cipher calls as for the standard encryption.

3. **Security Assumptions**

   - In [69], the authors set a doubt on the clarity and correctness of the security claims of the CCM mode and give some example attacks on the mode. Also, the nonce length in CCM is restricted in such a way that it may not provide adequate security when nonces are chosen randomly.

3.5.2.2 Extended message encrypted authentication mode: EAX

In [13] M. Bellare, P. Rogaway and D. Wagner introduce the EAX mode. The EAX is also a AEAD scheme based on the generic composition Encrypt- then-MAC. EAX uses CTR mode in encryption and the OMAC [48] message authentication code with a single key. EAX takes advantage of the OMAC which supports arbitrary message and tag lengths (unlike CCM). The user parameters needed here are the standard: block cipher and tag lengths. Generally, EAX tends to be much simpler but slightly less efficient than CCM.

The encryption under EAX mode of operation is shown in Fig. 3.8. It works by first encrypting the plaintext message part in CTR mode, using an authenticated arbitrary length nonce value for an initial counter. The tag is generated in a second pass, where the header and encrypted parts are authenticated by OMAC. OMAC algorithm is based on CBC-MAC hashing approach, but uses an additional padding method. The concatenation of the encrypted message and the tag forms the ciphertext. In decryption, EAX first separates the tag from the ciphertext and then verifies the newly calculated tag (over C) with the sent one. Finally, if the verification pass is successful, the encrypted part of the data is decrypted in CTR mode. From the EAX decryption we see that EAX is pure verify then decrypt design. Moreover, like CCM, EAX uses only the forward direction of the block cipher, so that hardware implementations do not need to implement the decryption functionality.

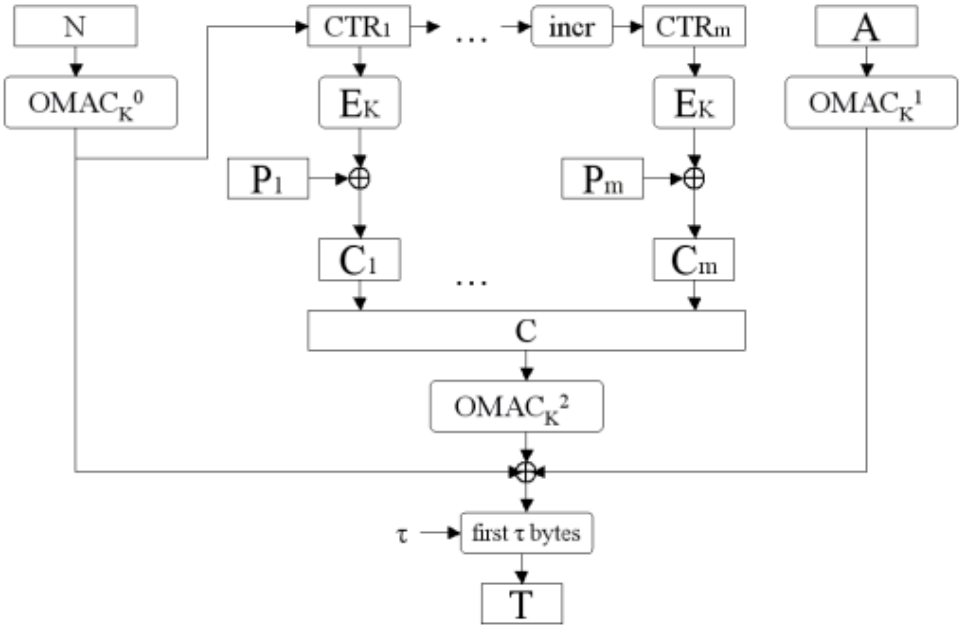


Figure 3.8: EAX mode of operation [13]. The plaintext is $P = P_1 \dots P_m$, the key is K and the associated data or header information is A. The ciphertext is $C||T$ where $C = C_1 \dots C_m$.

EAX properties

1. **General**
   - Single key usage.
   - It preprocess static headers or associate data. This reduces the cost of calculation, while CCM does not succeed in this.
   - Arbitrary length message, associated data and non-repeating nonces: arbitrary lengths are handled by both primitives of the EAX construction. Both CTR mode and OMAC operate on arbitrary long messages.
2. **Efficiency**
   - EAX, in contrast to CCM, is an online mode. It processes the message and the header with a constant memory, not knowing when the data stream will stop.
   - Fast verification: in decryption the favorable Verify then Decrypt order is followed by the receiver.
   - The major penalty on the efficiency of EAX, similarly to CCM, is the lack of parallel processing. This is caused by the OMAC usage that does not allow parallel processing;
   - Big number of block cipher calls : $2\lceil\frac{|P|}{bl}\rceil + \lceil\frac{|A|}{bl}\rceil + \lceil\frac{|N|}{bl}\rceil$ and $2\lceil\frac{|P|}{bl}\rceil + \lceil\frac{|N|}{bl}\rceil$ for the invocations with a static header where bl denotes the block length.
3. **Security**
   - EAX bases its security proofs on the pseudorandomness of the underlying block cipher (PRP) and achieves indistinguishability from random bits with respect to confidentiality and integrity of ciphertexts to authenticity.

3.5.2.3 CTR mode with Carter-Wegman authentication: CWC

In [54] T. Kohno, J. Viega and D. Whiting introduce the CWC mode, which is an AEAD scheme that uses CTR mode encryption, and instead of a block cipher based MAC it exploits the Carter-Wegman universal hashing for authentication.

Only one key K is used, but another one K_h is derived from K applying one block cipher invocation with the key K to the fixed constant. The result is converted from binary to integer number representation. At first, the unique nonce N is combined together with an incremental counter and used with the key K in CTR mode to encrypt the blocks $P_1, \ldots P_m$, resulting in the ciphertext $C = C_1, \ldots C_m$,. After then, an associated payload data A are bundled together with the ciphertext C, the length of the associated payload data $len(A)$ and a length of the ciphertext $len(C)$ and padded to the necessary length. The result Y is divided into 96-bits parts $Y_1, \ldots, Y_r, Y_{r+1}$. Each part is converted to an integer number and

used as coefficient for the Carter-Weigman polynomial[1] applied to value of K_h. Solving of the Carter-Weigman polynomial in a group over residual class with a prime base[2] can be done in parallel way. A whole process is shown in the Fig. 3.9. For a detailed specification one can refer to [54].

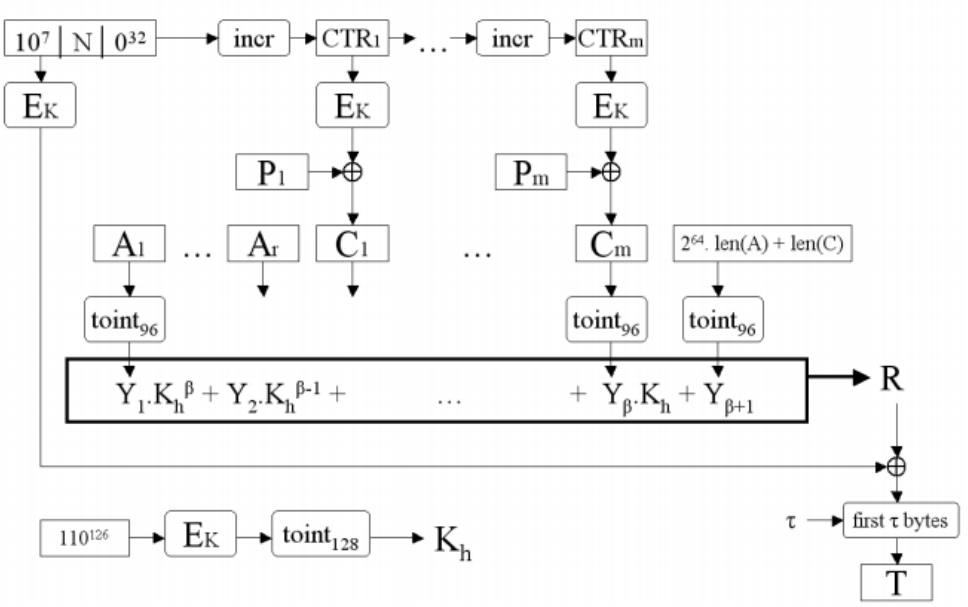


Figure 3.9: CWC mode of operation [73]. The plaintext is $P = P_1 \dots P_m$, the key is K and the associated data or header information is $A = A_1 \dots A_r$. The ciphertext is $C||T$ where $C = C_1 \dots C_m$.

CWC properties

1. **General**

   - Single key usage: CWC is referred to as one key mode, but it uses a subkey K_h, which can be either internally stored or derived when needed. The derivation process can be done across block cipher invocations. It is cheaper and requires a simplified key management, but an additional block cipher call.
   - CWC is capable of pre-processing static headers.
   - CWC cannot handle arbitrary length messages, the message M and the associated data A are a multiple of 8-bits, but this can easily be modified (according to the authors).

2. **Efficiency**

[1] $Y_1 x^r + Y_2 x^{r-1} + Y_3 x^{r-2} + \dots + Y_r x + Y_{r+1} mod 2^{127} - 1$
[2] $2^{127} - 1$ is a prime.

- Online processing: data is processed as it arrives.
- Parallelizable as CWC is a compound of parallelizable CTR mode and parallelizable hash function computation.
- Reduced number of block cipher calls compared to EAX and CCM. If the universal subkey K_h is computed across the block cipher invocations, the total number of block cipher calls is $\lceil \frac{|P|}{128} \rceil + 3$. If K_h is derived at the key generation stage, CWC requires one block cipher invocation at session setup stage and additional $\lceil \frac{|P|}{128} \rceil + 2$ invocations.

3. **Security**

   - CWC, like EAX, is provably secure. It assumes that underlying block cipher is a secure PRF. he CWC hash function family has to be AXU, such that the authenticity of CWC is proved correctly.

3.5.2.4 Galois Counter mode: GCM

The introduced by D. A. McGrew and J. Viega GCM [58] mode is one of the widely used AEAD modes. The mode uses CTR mode for encryption and universal hashing in the finite Galois Field (2^{128}), called Ghash, for message authentication. The difference between CWC and GCM is in the evaluation of universal hash function that both modes use. While CWC uses universal hashing based on integer multiplication in the GF $(2^{127} - 1)$, the GCM uses multiplication in the binary GF(2^{128}). he urge for such change is that binary field multiplication can be implemented easier in hardware and be made very efficient in software by using table-driven methods.

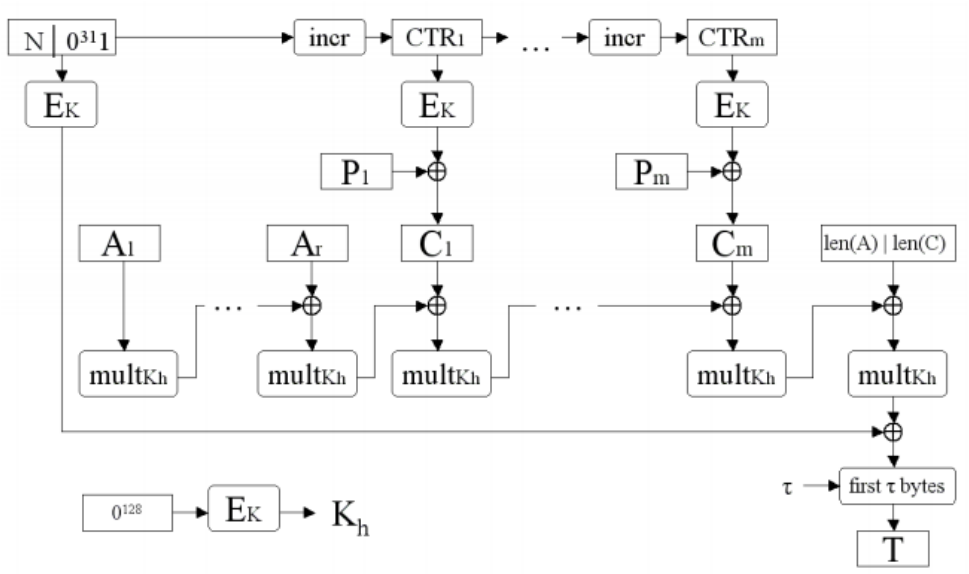


Figure 3.10: GCM mode. The plaintext is $P = P_1 \dots P_m$, the key is K and the associated data or header information is $A = A_1 \dots A_r$. The ciphertext is $C||T$ where $C = C_1 \dots C_m$.

The encryption and authentication process of GCM is illustrated on Fig. 3.10. GCM functions by first deriving a hash key K_h from K in key setup time. Then an arbitrary length nonce N is encrypted, if it is of size 96- bits, otherwise N is preprocessed by Ghash before its encryption. Next, an arbitrary length plaintext P is processed in blocks in CTR mode, using incremented N for initial counter, to get the corresponding blocks of C. The header A and C are then Ghash-ed and the resulting hash value is XOR-ed with the preprocessed N value to get the tag T. The final tag length is then the fixed by the users input most significant t bits, $t \in \{0, ..., 128\}$ of T. At the end, the chopped tag is appended to C to generate the final ciphertext. In GCM decryption, the order of Verify then Decrypt is followed. For more detailed description, one can refer to [58].

GCM properties

1. **General**

   - Usage of single key: although internally GCM uses second K_h hashing subkey, which is derived in session setup time usually, it is considered to be a single key mode like CWC.
   - Preprocessing of the keystream, static headers or fixed parts of the nonce.
   - Nonce, associated data and message arbitrary length. GCM and EAX allow arbitrary length nonces, which makes them better.

2. **Efficiency**

- Online processing: data is processed as it arrives.
- Parallelizable as GCM is a compound of parallelizable CTR mode and parallelizable hash function computation.
- Very good hardware and software performance compared with the other conventional modes, due to the usage of hash function evaluation in GF(2^{128}) for authentication.
- Minimum number of block cipher calls: $\lceil \frac{|P|}{128} \rceil + 1$ and (+1) if K_h is not stored, but derived across block cipher invocations.

3. **Security**

- The security proofs are based on the pseudorandomness of the unde lying block cipher. The confidentiality is proven as indistinguishability of ciphertext from random bits under chosen plaintext attack. For the authenticity hash function family, similar to CWC, needs to be AXU to achieve authenticity.

3.6 Dedicated schemes

The 'generic composition' approach above is not the right answer for everyone. It requires to implement two different primitives (say, a block cipher and a MAC), which can be a hassle. And, it isn't necessarily the fastest way to get messages encrypted.

For all of these reasons, we have specialized block cipher modes of operation called Authenticated Encryption modes. These modes handle both the encryption and the authentication in one go, usually with a single key. IAPM, OCB and SpongeWrap are well known example in this category. We will explain all of these modes briefly here.

3.6.1 Integrity aware parallelizable mode: IAPM

The first proposal for AE schemes that manage to merge the encryption and authentication passes into one are Jutla's IAPM mode [52]. Jutla's idea is a breakthrough in the state of AE modes, though relatively simple. IAPM mode [52] proposed by Ch. Jutla is a first provable secure mode for authenticated encryption. It requires two independent keys K_0, K_1 with the same length as an underlying block cipher. Both an encryption and a decryption direction of a block cipher are used. An unique nonce N (for each message using same keys K_0, K_1) is used to generate an offsets, a pair wise differentially uniform vectors[3] with $m+1$ members. This generation requires a one block cipher invocation using a key K_0, an integer additions over the Galois field using n bits prime p (GF_p) respectively only by the xor operations with a penalty of approximately log_m extra block cipher invocations. The nonce N is communicated as part of a ciphertext. The rest of encryption process is shown

[3] A sequence of uniformly distributed n-bit random numbers $S_1, S_2, \ldots, S_m$, is called pair-wise independent if for every pair $i, j, i \neq j$, and every pair of n bit constants c_1 and c_2, a probability that $S_i = c_1$ and $S_j = c_2$ is 2^{-2n}. In other words, S_i is random and has no special relation to S_j .

in Fig. 3.11 and consists m block cipher invocations using key K_1 and the integer additions over GF_p. Checksum denotes to $P_1 \oplus P_2 \oplus \ldots \oplus P_{m1}$. During the decryption process, same offsets using K_0 are generated, and the encryption process is simply reversed. After the decryption, message integrity is verified by checking if $P_m = P_1 \oplus P_2 \oplus \ldots \oplus P_{m1}$. For the detailed specification one can refer to [52].

Jutla also proposed another one mode called Integrity Aware Cipher Block Chaining Mode (IACBC) [52]. This mode is like IAPM, but instead of xoring the offset S_i to P_i (before block cipher invocation), ciphertext of the previous block is used (as in the CBC mode).

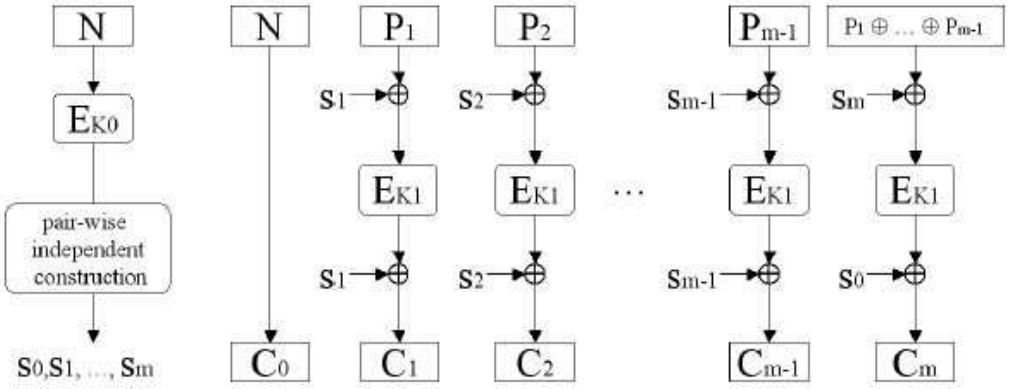


Figure 3.11: Integrity Aware Parallelizable Mode scheme

IAPM properties

1. **General**
   - The main drawback of IAPM is the usage of 2 keys K_0 and K_1.
   - IAPM can not process arbitrary length messages, the messages have to be a multiple of the block length.
   - IAPM does not provide support for associated data.
2. **Efficiency**
   - Online processing: data is processed as it arrives;
   - IAPM is based on the ECB construction and is therefore parallelizable;
   - Low cost : for long messages the cost of the IAPM processing is almost same as the CBC or the CTR mode encryption of the data.
3. **Security**
   - IAPM is provably secure.

3.6.2 Offset CodeBook mode : OCB

Offset CodeBook mode (OCB) [67] proposed by Rogaway et. al is based on IAPM mode, but some new characteristics are added. Only one underlying key K is needed for the offset calculations and the data encryption. More efficient way for the offset calculations is employed. Message integrity is verified using authentication tag T with optional length τ (up to n-bits), varying depending on an application needs. Initial version of OCB does not provide support for the associate data, however later version of OCB introduced in [66] provide the support for same.

Both the encryption and the decryption direction of the block cipher are used. For every new message, non-repetitive nonce is used. For the key value K value $L \leftarrow E_K(0^n)$ is computed (only once if L can be stored for a future invocations). For each new nonce N, one block cipher invocation is used to create the intermediate value $R \leftarrow E_K(N \oplus L)$. Using the Gray codes γ, L and R, the offsets Z_i are generated $Z_i \leftarrow \gamma_i.L \oplus R$. The rest of the encryption process is shown in the Fig. 3.12 and consists m block cipher invocations using the key K and the xor operations. Checksum denotes to $P_1 \oplus P_2 \oplus \ldots \oplus P_{m1} \oplus C_m 0^* \oplus Y_m$. For the detailed specification one can refer to [67].

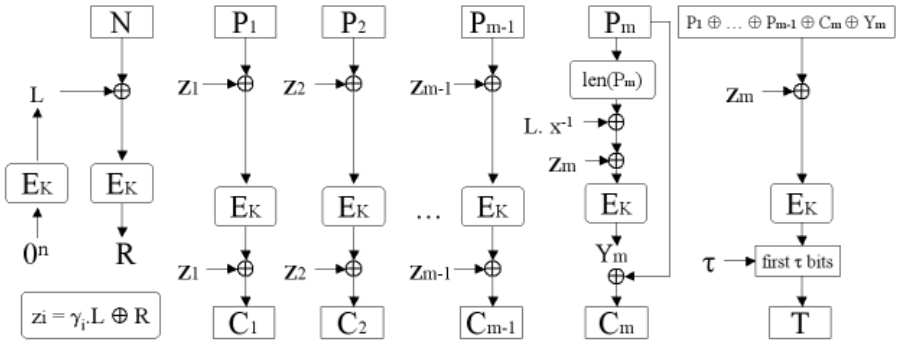


Figure 3.12: Offset CodeBook Mode scheme

OCB properties

1. **General**

   - Single key usage.
   - IOCB processes the last message block in a different way than IAPM, so that arbitrary length messages can be processed. Notice that when the message is a multiple of the block length, then the last block is processed as in IAPM mode.
   - OCB does not provide support for associated data.
   - Nonce repetition is not allowed.

2. **Efficiency**

   - Online processing: data is processed as it arrives.
   - Fully parallelizable, like IAPM, OCB is based on the ECB parallelizable construction.
   - OCB, like IAPM, does not succeed in fast verification. In decryption, the ciphertext needs to be first received and then operation on the data is permitted. This is a considered disadvantage of the mode.
   - Reduced number of block cipher calls: $\lceil \frac{|M|}{bl} \rceil + 2$, where bl is block length, as OCB makes invocations for plaintext encryption, plus one for randomizing the checksum and one encryption of the nonce.

3. **Security**

   - OCB achieves indistinguishability from random under CCA with respect to privacy. Because of the use of a randomized checksum for authentication, the mode has stronger requirements for the encryption function than being PRP, like IAPM, the block cipher has to be secure PRP under CCA. This, as pointed by Rogaway assures a good authentication.

3.6.3 SpongeWrap

SpongeWrap [14] mode is based on the Sponge construction and developed by Bertoni et. al. This is the first AE mode which is based on permutation. The security of this construction is based on the security of Sponge construction which is provably secure, thus making SpongeWrap a secure AE mode. It supports AEAD and it can also give intermediate tag for the plaintext if required. Further, variable length tag is also possible in this scheme. It requires m permutation calls for an m block plaintext, including associated Data. SpongeWrap can be used for key wrapping (sending the cryptographic keys through open channel).

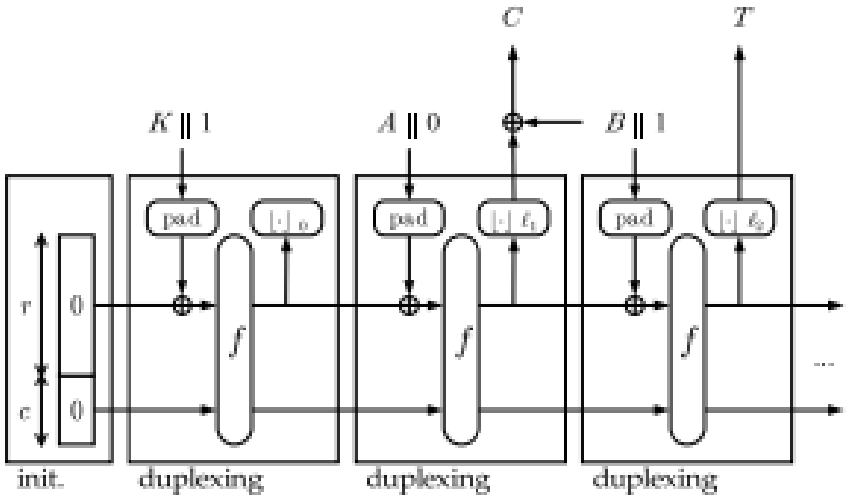


Figure 3.13: SpongeWrap mode

3.7 Comparison of AE schemes

In this section, we summarize the properties of all the AE modes discussed above in this chapter. Table 3.7 presents the summary of all the AE modes.

Mode	**CCM**	**EAX**	**CWC**	**GCM**	**IAPM**	**OCB**
AE/AEAD	AEAD	AEAD	AEAD	AEAD	AE	AE
Pre-process static AD	No	Yes	Yes	Yes	-	-
N/IV length	fixed N 56-104 bits	not fixed N *	fixed N 88 bits	not fixed N *	not fixed N/IV	fixed N *bl* bits
No. of keys	1	1	1	1	2	1
Message length	multiple of 8 bits	arbitrary	multiple of 8 bits	arbitrary	multiple of block length	arbitrary
No. of block cipher calls	$2.P_{bl} + A_{bl} + 2 + \delta$	$2.P_{bl} + A_{bl} + N_{bl}$	$P_{bl} + 2$	$P_{bl} + 1$	$P_{bl} + 2 + log(n+1)$	$P_{bl} + 2$
Block cipher calls in setup	0	0	1	1	0	1
Parallelizable	No	No	Yes	Yes	Yes	yes
Online	No	Yes	Yes	Yes	Yes	Yes
Encryption	CTR	CTR	CTR	CTR	rand. ECB	rand. ECB
Authentication	CBC-MAC	OMAC	Uhash $GF(2^{127}-1)$	Uhash $GF(2^{128})$	rand. + Enc-XOR-Sum	rand. + Enc-XOR-Sum
Verify-then-Decrypt/ Decrypt-then-Verify	Verify-then-Decrypt	Verify-then-Decrypt	Verify-then-Decrypt	Verify-then-Decrypt	Decrypt-then-Verify	Decrypt-then-Verify
Provably secure	Yes	Yes	Yes	Yes	Yes	Yes
Patent-free	Yes	Yes	Yes	Yes	No	No

Table 3.2: Basic properties of the AE schemes: CCM, EAX, CWC, GCM, IAPM and OCB.

3.8 Security Notions for Authenticated Encryption

The two main security notions of the AE is privacy and authenticity of ciphertext and tag. These are the basic requirement for any AE schemes.

3.8.1 Privacy

The privacy of the encryption scheme can be proved by showing the proposed scheme is indistinguishable from a Random Oracle. The security proof is given based on the advantage of an adversary to distinguish between the Encryption oracle and the Random Oracle.

Consider an adversary $\mathcal{A}$ who is made to interact with either the real encryption oracle $\mathcal{E}nc_K(N, M)$ or an ideal function $\$(N, M)$. The advantage of the adversary is the ability to identify with which it is interacting (shown in Fig. 3.14). We then state that the scheme achieves CPA privacy in the indistinguishability from random strings sense, if the adversarial advantage is negligible. If the underlying primitive is ideal random function, say ro, then ro is made available to the adversary $\mathcal{A}$ in both the cases. If the underlying primitive is ideal permutation, then in either of these two exclusive cases, an ideal permutation π and its inverse permutation π^{-1} is available to $\mathcal{A}$. The advantage of the adversary is the ability to identify with which it is interacting. This is given by IND-CPA advantage, which is defined as follows.

Definition 3. ***(IND-CPA advantage)*** *For a given authenticated encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}nc, \mathcal{D}ec)$, IND-CPA advantage is defined as the ability of an adversary $\mathcal{A}$ to distinguish between the output of an encryption oracle $\mathcal{E}$ from the random function $\$$. Let $\mathcal{A}$ be computationally bounded adversary with access to an oracle $\mathcal{O}$, Then IND-CPA advantage of $\mathcal{A}$ relative to $\mathcal{AE}$ is given as:*

$$\begin{aligned} \textit{For ideal permutation}, \mathrm{IND-CPA}_{\Pi}(\mathcal{A}) &= \Delta(\mathcal{E}_K, \pi, \pi^{-1}; \$, \pi, \pi^{-1}) \\ \textit{For ideal random function}, \mathrm{IND-CPA}_{\Pi}(\mathcal{A}) &= \Delta(\mathcal{E}_K, ro; \$, ro) \end{aligned}$$

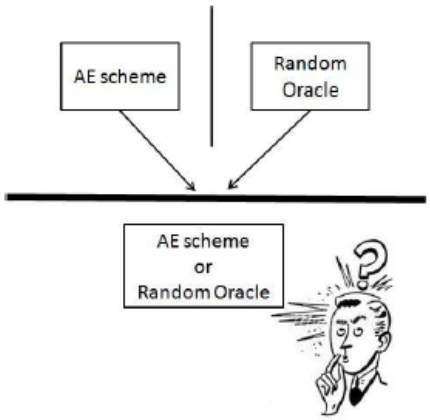


Figure 3.14: Indistinguishability framework for Privacy: Adversary has to find out whether it is interacting with AE scheme or a random oracle.

If the scheme uses nonce, then the adversary $\mathcal{A}$ is nonce-respecting, i.e. it is not allowed to repeat the nonce. That is, if $\mathcal{A}$ asks the oracle query(N, M) then it will never ask the oracle with another query(N, M') where $M \neq M'$.

3.8.2 Authenticity

The authenticity of an authenticated encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}nc, \mathcal{D}ec)$, is defined in terms of the ability of an adversary who can produce a valid (N, C, T) pair without querying the oracle for that corresponding M.

The following experiment $Exp_{\Pi}^{auth}(\mathcal{A})$ is carried out to provide a bound for authenticity. The forging adversary $\mathcal{A}$ is given complete access to encryption oracle $\mathcal{E}nc_K(.,.)$, decryption oracle $\mathcal{D}ec_K(.,.,)$,either ro or (π and π^{-1}). Further, $\mathcal{A}$ can query these oracles for some fixed number of times. $\mathcal{A}$ must be nonce-respecting with encryption queries. Finally, it must output (N, C, T). We say $\mathcal{A}$ forges the AE scheme if for the given (N, C, T), the decryption algorithm outputs a valid M and the tuple (N, C, T) has not already been output by $\mathcal{E}nc_K()$.

Definition 4. ***(Authenticity)*** *The advantage of the Adversary $\mathcal{A}$ in forging a given authenticated encryption scheme Π is represented as*

$$Adv_{\Pi}^{auth}(\mathcal{A}) = Pr[Exp_{\Pi}^{auth}(\mathcal{A}) = 1].$$

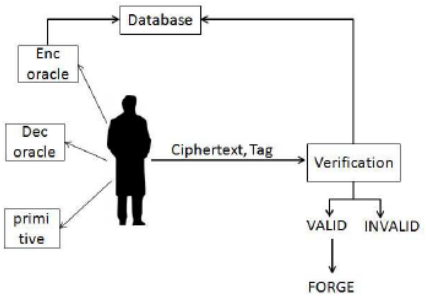


Figure 3.15: Authenticity experiment: Adversary has to forge ciphertext, tag pair. Adversary has access to encryption, decryption and primitive oracles to forge.

3.9 Various other Security Notions

Some recent papers introduced new security notions to strengthened AE. These new security notions include releasing unverified plaintext [35], and robust authenticated encryption [44]. We will discuss each of them in the following sections.

3.9.1 Releasing Unverified Plaintext (RUP)

In [35], Andreeva et al. proposed a new decryption model that release a plaintext before confirming its validity. In practice, such a candidate plaintext often becomes available (including to an adversary), even if validation fails. Examples include all schemes that need to decrypt before integrity can be checked as well as schemes sporting online decryption.

The RUP paper separate the actual decryption oracle in conventional AE scheme into two separate oracles. One is decryption oracle $\mathcal{D}ec$ that always prepares a purported plaintext and a verification oracle $\mathcal{V}er$ that determines whether the ciphertext was valid or not. The idea is that in honest implementations the output of $\mathcal{D}ec$ is only released if verification passes, yet security should hold even if the output of $\mathcal{D}ec$ is leaked before verification takes place (corresponding to releasing unverified plaintext). With this modification, the RUP algorithms $\mathcal{D}ec$ and $\mathcal{V}er$ satisfy

$$\begin{aligned} \mathcal{D}ec &: \mathtt{K} \times \mathtt{N} \times \mathtt{A} \times \mathtt{C} \to \mathtt{M} \\ \mathcal{V}er &: \mathtt{K} \times \mathtt{N} \times \mathtt{A} \times \mathtt{C} \to \{\top/\bot\} \end{aligned}$$

When called with a valid ciphertext, $\mathcal{D}ec_k$ returns the plaintext, and $\mathcal{V}er_k$ returns $\top$. Conversely, when called with an invalid ciphertext, $\mathcal{D}ec_k$ will return some leakage information (unverified plaintext) and $\mathcal{V}er_k$ will return $\bot$.

In this paper, they provided a large number of new definitions, covering security under this new notion for both confidentiality and integrity.

Security Notions: In addition to standard AE security notions, IND-CPA and INT-CTXT, Andreeva et. al provided 2 new security notions called PA (Plaintext Awareness) and INT-RUP (Integrity under Releasing Unverified Plaintext). To provide privacy PA is used along with IND-CPA. At the heart of PA notion is plaintext extractor. An encryption scheme is said to be PA if there exists an efficient plaintext extractor for every adversary that mimicks the decryption oracle in order to fool an adversary. Plaintext extractor is not allowed to make the encryption nor decryption queries and does not even have information about the secret key. There are two notions of plaintext awareness: PA1 and PA2. In PA1 settings, extractor is given access to the history of queries made to encryption oracle, but not in PA2. Hence, PA2 is more stronger notion than PA1. An AE scheme is INT-RUP if adversary can not find a new ciphertext tag pair which passes the verification, with having access to the encryption and unverified decryption oracle.

Definition 5. ***(PA1 advantage)*** *Let* $\Pi = (\mathcal{E}, \mathcal{D})$ *be an encryption scheme and* $\mathcal{A}$ *be an adversary with access to two oracles* $\mathcal{O}_1$ *and* $\mathcal{O}_2$*. Let* $\mathtt{E}$ *be an algorithm with access to the history of queries made to* $\mathcal{O}_1$ *by* $\mathcal{A}$*, called a PA1- extractor. We allow* $\mathtt{E}$ *to maintain state across invocations. The PA1 advantage of* $\mathcal{A}$ *relative to* $\mathtt{E}$ *and* Π *is given as following:*

$$\mathrm{PA1}^{\mathtt{E}}_{\Pi}(\mathcal{A}) = \Delta(\mathcal{E}_{\mathrm{K}}, \mathcal{D}_{\mathrm{K}}; \mathcal{E}_{\mathrm{K}}, \mathtt{E})$$

The adversary $\mathcal{A}$ tries to distinguish the case in which its second oracle $\mathcal{O}_2$is given by $\mathcal{D}_K$ versus the case in which $\mathcal{O}_2$ is given by $\mathtt{E}$. The task of $\mathtt{E}$ is to mimic the outputs of $\mathcal{D}_K$ given only the history of queries made to $\mathcal{D}_K$ by $\mathcal{A}$ (the key is not given to $\mathtt{E}$). Note that $\mathcal{A}$ is allowed to make queries of the form $\mathcal{E}_K \rightarrow \mathtt{E}$; these can easily be answered by $\mathtt{E}$ via the query history.

PA2 is a strengthening of PA1 where the extractor no longer has access to the query history of $\mathcal{E}_K$.

Definition 6. ***(PA2 advantage)*** *Let* $\mathcal{A}$ *be an adversary as in Def. 5, with the added restriction that it may not ask queries of the form* $\mathcal{O}_1 \leftarrow \mathcal{O}_2$*. Let* $\mathtt{E}$ *be an algorithm, called a PA2-extractor. We allow* $\mathtt{E}$ *to maintain state across invocations. The PA2 advantage of* $\mathcal{A}$ *relative to* $\mathtt{E}$ *and* Π *is*

$$\mathrm{PA2}^{\mathtt{E}}_{\Pi}(\mathcal{A}) = \Delta(\mathcal{E}_{\mathrm{K}}, \mathcal{D}_{\mathrm{K}}; \mathcal{E}_{\mathrm{K}}, \mathtt{E})$$

INT-RUP Under INT-CTXT, adversary goal is to produce a new valid (C, T) pair with only having access to the encryption oracle. In RUP setting INT-CTXT is translated into INT-RUP, by allowing the adversary to observe unverified plaintext. This is achieved by separating decryption oracle from verification, and giving the adversary access to a decryption oracle $\mathcal{D}_{\mathcal{K}}$.

Definition 7. *Let* $\Pi = (\mathcal{E}, \mathcal{D}, \mathcal{V})$ *be an authenticated encryption scheme with separate decryption and verification oracle and* $\mathcal{A}$ *be a computationally bounded adversary having access to* $\mathcal{E}_K, \mathcal{D}_K$ *and* $\mathcal{V}_K$ *oracles for a fixed key* K*, such that* $\mathcal{A}$ *never makes a queries of*

type $\mathcal{E}_K \hookrightarrow \mathcal{V}_K$. *Then the INT-RUP advantage of adversary* $\mathcal{A}$ *corresponding to* Π *is given as following:*

$$\mathrm{INT} - \mathrm{RUP}_{\Pi}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathcal{E}_{\mathrm{K}},\mathcal{D}_{\mathrm{K}},\mathcal{V}_{\mathrm{K}}}\mathrm{Forges}].$$

3.9.2 Robust Authenticated Encryption (RAE)

Hoang et. al proposed a robust authenticated encryption in [44]. With a scheme for robust authenticated encryption a user can select an arbitrary value $\lambda \geq 0$ and then encrypt a plaintext of any length into a ciphertext thats λ characters longer. All values $\lambda \in \mathbb{N}$ should be allowed. It might be OK to set some reasonable upperbound $\lambda \leq \lambda_{max}$, but there should not be a nonzero lowerbound.The scheme must provide all the privacy and authenticity possible for the requested λ. To achieve this, both encryption and decryption algorithms are provided with an additional input, called the stretch parameter λ , leading to the syntax:

$$\begin{aligned} \mathcal{E}nc &: \mathtt{K} \times \mathtt{N} \times \mathtt{A} \times \mathbb{N} \times \mathtt{M} \to \mathtt{C} \\ \mathcal{D}ec &: \mathtt{K} \times \mathtt{N} \times \mathtt{A} \times \mathbb{N} \times \mathtt{C} \to M/\perp \end{aligned}$$

Fix an alphabet Σ. RAE scheme can be defined as a triplet $\Pi = (\mathcal{K}, \mathcal{E}nc, \mathcal{D}ec)$ but with the signature of $\mathcal{E}nc$ and $\mathcal{D}ec$ updated. The encryption algorithm $\mathcal{E}nc$ is deterministic and maps a five-tuple $(K, N, A, \lambda, M) \in (\Sigma^*)^3 \times \mathbb{N} \times \Sigma^*$ to a string $C = \mathcal{E}nc_k^{N,A,\lambda}(M)$ of length $|M| + \lambda$. For maximal utility when realized, we are not permitting a return value of $\perp$: an RAE scheme must be able to encrypt any M using any N, A, and λ. The decryption algorithm $\mathcal{D}ec$ is deterministic and takes a five tuple (K, N, A, λ, C) to a value $\mathcal{D}ec_k^{N,A,\lambda}(C) \in \Sigma^* \cup \{\perp\}$. We require that $\mathcal{D}ec_k^{N,A,\lambda}(\mathcal{E}nc_k^{N,A,\lambda}(M)) = M$ for all K, N, A, λ, M. If there is no M such that $C = \mathcal{E}nc_k^{N,A,\lambda}(M)$ then $\mathcal{D}ec_k^{N,A,\lambda}(C) = \perp$.

Chapter 4

Preliminaries

4.1 Provable Security

Whenever we propose a cryptosystem, we should provide some indication that it is secure. There are lots of ways to provide evidence that a scheme is secure. None of them guarantee security. Provable security aims to give a mathematical guarantee that a scheme is secure. The modern notion of provable security of cryptographic schemes was first introduced by Goldwasser and Micali in [39]. Informally, a scheme is provably secure if it comes with a rigorous logical argument that shows that if the security of this scheme is compromised then

- either some simple logical contradiction occurs (Information theoretic security or security against computationally unbounded adversaries),
- or some well-studied problem can be solved efficiently (security against computationally bounded adversaries).

In the latter case, one must first assume the hardness of some problem (such as factorization of large integers) or the existence of some primitive (such as a one-way function f, for which it is easy to compute $f(x)$, but given $y = f(x)$ it is computationally intractable to recover$x$). In order to prove the security of a cryptographic scheme, one shows that a potential adversary against the scheme (i.e., an algorithm that breaks the scheme) can be used as a subroutine in order to efficiently break the computational assumption. We say that the cryptographic scheme reduces to the computational assumption.

For giving a security proof of a given scheme, we need

- a definition of the scheme
- a definition of security, including an execution environment / model and a winning condition
- a class of attackers
- a proof that no attacker in that class can achieve the winning condition for that scheme

Provable security has several approaches as shown in fig. 4.1

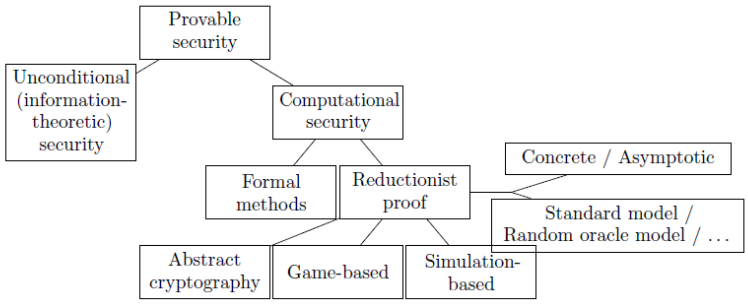


Figure 4.1: Provable Security approaches

4.2 Adversary Models

Security proof of any cryptographic scheme starts with defining the adversarial model under which the protocol is secure. The assumptions of underlying primitives, the adversary's power and its limitation are defined in these models. There are three main models available. a) Standard Model; b) Random Oracle Model and c) Ideal Cipher Model.

4.2.1 Standard Model

The standard model is the model of computation in which the adversary is only limited by the amount of time and computational power available. In this model the assumptions are based on well studied problems like factoring the product of two large primes, discrete logarithmic problem etc. Schemes which can be proven secure under these assumptions are said to be secure in the standard model. Security proofs are notoriously difficult to achieve in the standard model, so in many proofs, cryptographic primitives are replaced by idealized versions. this leads to random oracle model.

4.2.2 Random Oracle Model

Bellare and Rogaway formally introduced Random Oracle Model in [10]. It is a widely used model to provide the security of cryptographic schemes. In this model, there is a publicly accessible random function which takes $\{0, a\}^n$ and outputs n bits and this output is uniform and independent of other outputs. The public function is called Random Oracle, which we will define in details in Section 4.3.

4.2.3 Ideal Cipher Model

The Ideal Cipher Model is another idealized model of computation, similar to the Random oracle model. Instead of having a publicly accessible random function, one has a publicly

accessible random block cipher (or ideal cipher). This is a block cipher with a k-bit key and a n-bit input/output, that is chosen uniformly at random among all block ciphers of this form; this is equivalent to having a family of 2^k independent random permutations.

The ideal cipher model is similar to random oracle model with certain exceptions [19].

- Ideal cipher need to be PRP where as a random oracle is a random function.
- Adversaries interacting in Ideal Cipher Model have access to the cipher and its inverse whereas in Random Oracle Model adversary does not get access to the inverse of the random function.
- Ideal cipher can take n bit input only whereas random oracle can take infinite bits as input.

4.3 Random Oracle

A random oracle(RO) is described by the following model:

- There is a black box. In the box lives a gnome, with a big book and some dice.
- We can input some data into the box (an arbitrary sequence of bits).
- Given some input that he did not see beforehand, the gnome uses his dice to generate a new output, uniformly and randomly, in some conventional space (the space of oracle outputs). The gnome also writes down the input and the newly generated output in his book.
- If given an already seen input, the gnome uses his book to recover the output he returned the last time, and returns it again.

RO is written as $\mathcal{R} : \{0,1\}^* \leftarrow \{0,1\}^n$. Let this RO be queried for inputs $X_1, X_2 \dots X_q$. One of the key properties of an RO is that the output to the i^{th} query is independent of the answers it has produced for earlier queries. That is, for $Y_k \in \{0,1\}^n$ for any k and $X_i \neq X_j : j < i$, we have that

$$Pr[(\mathcal{R}(X_i) = Y_i)|(\mathcal{R}(X_1) = Y_1) \wedge (\mathcal{R}(X_1) = Y_1) \dots (\mathcal{R}(X_{i-1}) = Y_{i-1})] = 2^{-n} \tag{4.1}$$

4.4 Proof Techniques

Privacy and authenticity are proved using certain tools or in other words, it is called Framework. There are major two techniques that are used to write a security proof: Game playing and the H-coefficient technique. In this section, we have given the basics for both of those techniques. However, in this thesis, we have only used the Game Playing technique to provide the privacy and authenticity of the proposed schemes.

4.4.1 Game Playing Framework

The idea here is to represent a protocol/primitive as a game played between an attacker and a challenger. We define a bad event and show through the game that the event happens with (close to) some target probability (say 1/2 or 0). For example, if we are modeling an RNG and want to show that the attacker can only guess the next output bit with probability very close to (formally, negligibly close to) 1/2. If they are able to do better than that, then the RNG isn't very good. 1/2 is chosen since there are only 2 choices for the next bit (0 or 1) and they should occur each with probability approximately 1/2 so an attacker who simply always guesses 0, for example, should be right 50% of the time.

Game playing technique proposed by Bellare and Rogaway in [12]. The main purpose of this framework is to find out the ability of an adversary to distinguish a actual protocol/primitive and a random one. Using this framework, we start with the definition of the proposed scheme and modify the scheme little by little till the scheme uses a random function. While modifying the games, the probability difference between the two games are calculated and finally all the probabilities are summed up to give the advantage of the adversary to distinguish the proposed scheme and the random function.

$$Adv_{\Pi}^{priv}(\mathcal{A}) = |Pr[\mathcal{A}^{First\ game}] - Pr[\mathcal{A}^{Final\ game}]|$$

Games are like a program. It follows pseudo code language. A game consist of three procedures a) Initialize; b) Finalize and c) Named Oracles (each one a procedure).

To prove security using the approach, one proceeds as follows [72]. One constructs a sequence of games, Game 0, Game 1, . . . , Game n, where Game 0 is the original attack game with respect to a given adversary and cryptographic primitive. Let S_0 be the event S, and for$i = 1, \ldots, n$, the construction defines an event S_i in Game i, usually in a way naturally related to the definition of S. The proof shows that $Pr[S_i]$ is negligibly close to $Pr[S_i + 1]$ for $i = 0, \ldots, n - 1$, and that $Pr[S_n]$ is equal (or negligibly close) to the target probability. From this, and the fact that n is a constant, it follows that $Pr[S]$ is negligibly close to the target probability, and security is proved.

This is the general framework of such a proof. However, in constructing such proofs, it is desirable that the changes between successive games are very small, so that analyzing the change is as simple as possible. In general, these transitions between games can be restricted to one of the three types.

Transitions based on indistinguishability : In such a transition, a small change is made that, if detected by the adversary, would imply an efficient method of distinguishing between two distributions that are indistinguishable. For example, suppose P1 and P2 are assumed to be computationally indistinguishable distributions. To prove that $|Pr[S_i] - Pr[S_{i+1}]|$ is negligible, one argues that there exists a distinguishing algorithm D that interpolates between Game i and Game $i + 1$, so that when given an element drawn from distribution P_1 as input, D outputs 1 with probability $Pr[S_i]$, and when given an element drawn from distributionP_2 as input, D outputs 1 with probability $Pr[S_{i+1}]$. The indistinguishability assumption then implies that $|Pr[S_i] - Pr[S_{i+1}]|$ is negligible. Usually,

the construction of D is obvious, provided the changes made in the transition are minimal.

Transitions based on bad events: In such a transition, one argues that Games i and $i+1$ proceed identically unless a certain bad event E occurs. To make this type of argument as cleanly as possible, it is best if the two games are defined on the same underlying probability space, the only differences between the two games are the rules for computing certain random variables. When done this way, saying that the two games proceed identically unless E occurs is equivalent to saying that

$$S_i \wedge \neg \mathrm{E} \Longleftrightarrow S_{i+1} \wedge \neg \mathrm{E}$$

that is, the events $S_i \wedge \neg \mathrm{E}$ and $S_{i+1} \wedge \neg \mathrm{E}$ are same. If this is true, then we can use the following lemma, which is completely trivial, yet is so often used in these types of proofs.

Lemma 1. *Let* $\mathrm{A}, \mathrm{B}, \mathrm{E}$ *be events defined in some probability distribution, and suppose that* $\mathrm{A} \wedge \neg \mathrm{E} \Longleftrightarrow \mathrm{B} \wedge \neg \mathrm{E}$. *Then* $|Pr[\mathrm{A}] - Pr[\mathrm{B}]| \leq Pr[\mathrm{E}]$.

Proof. This is a simple calculation. We have

$$\begin{aligned} |Pr[\mathrm{A}] - Pr[\mathrm{B}]| &= |Pr[\mathrm{A} \wedge \mathrm{E}] + Pr[\mathrm{A} \wedge \neg \mathrm{E}] - Pr[\mathrm{B} \wedge \mathrm{E}] - Pr[\mathrm{B} \wedge \neg \mathrm{E}]| \\ &= |Pr[\mathrm{A} \wedge \mathrm{E}] - Pr[\mathrm{B} \wedge \mathrm{E}]| \\ &\leq Pr[\mathrm{E}] \end{aligned}$$

The second equality follows from the assumption that $\mathrm{A} \wedge \neg \mathrm{E} \Longleftrightarrow \mathrm{B} \wedge \neg \mathrm{E}$, and so in particular, $\mathrm{A} \wedge \neg \mathrm{E} = \mathrm{B} \wedge \neg \mathrm{E}$. The final inequality follows from the fact that both $Pr[\mathrm{A} \wedge \mathrm{E}]$ and $Pr[\mathrm{B} \wedge \mathrm{E}]$ are numbers between 0 and $Pr[\mathrm{E}]$. □

Bridging steps: The third type of transition introduces a bridging step, which is typically a way of restating how certain quantities can be computed in a completely equivalent way.The change is purely conceptual, and $Pr[S_i] = Pr[S_{i+1}]$. The reason for doing this is to prepare the ground for a transition of one of the above two types. While in principle, such a bridging step may seem unnecessary, without it, the proof would be much harder to follow.

4.4.2 Coefficients H Technique

In this section we discuss a powerful tool, called Coefficients H technique [61], which is used for bounding the distinguishing advantage of two random systems with respect to an adaptive adversary. The general setting of Coefficient H technique is as follows : We consider a adaptive adversary $\mathcal{A}$ that interacts with either of the two oracles : (i) real world oracle which is basically the cryptographic construction that we want to argue for its security and other is the ideal world oracle, for q many times where q is considered to be a parameter. Let $\mathcal{O}$ denotes the oracle to which adversary $\mathcal{A}$ interacts. That means $\mathcal{A}$ asks q many queries x to the oracle $\mathcal{O}$ and obtains the corresponding responses $y = \mathcal{O}(x)$ from

the oracle. Since we consider the adaptive adversary, the i-th query of $\mathcal{A}$ (i.e x^i) is some function of the previous $i1$ query responses. After such an interaction, $\mathcal{A}$ obtains a history of the query-responses $\tau = \{(x^1, y^1), (x^2, y^2), \ldots (x^q, y^q)\}$. This history of query-response that $\mathcal{A}$ obtains after the interaction to the oracle is over, is called the transcript of the adversary $\mathcal{A}$. Based on this transcript $\mathcal{A}$ outputs the final decision which is a deterministic function of the transcript obtained.

Now our goal is to find out the probability of realizing τ in real world and in ideal world separately. Note that, the motivation of introducing Coefficient H technique is to distinguish the output distribution of two random systems which essentially means that either of the oracles (real or ideal) to which the adversary interacts is a random system. This says that the real oracle $\mathcal{O}_{re}$ has its own randomness and the ideal oracle $\mathcal{O}_{id}$has its own source of randomness. In general, the underlying source of randomness in real oracle comes from choosing the key k of the cryptographic construction uniformly at random from the finite set of all keys K and some random coins (if any, random string is used in the construction besides of the key). Whereas the underlying source of randomness in ideal oracle is basically the random functions or the random permutations. Thus, it makes a perfect sense to talk about the probability of realizing a fixed transcript τ in the real and ideal oracle. Let X_{re} be the probability distribution of realizing a fixed transcript in real world. Similarly, X_{id} be the probability distribution of realizing a fixed transcript in ideal world with respect to the fixed adaptive deterministic distinguisher $\mathcal{A}$. Let $\mathcal{T}$ denotes the set of all transcripts τ. Then the advantage of $\mathcal{A}$ in distinguishing the output distribution of the two random system can be easily upper bounded by the statistical distance of X_{re} and X_{id}.

$$\Delta(X_{re}, X_{id}) = \frac{1}{2} \sum_{\tau \in \mathcal{T}} |Pr[X_{re} = \tau] - Pr[X_{id} = \tau]|.$$

The central ideal of the technique is to use the following lemma to bound the distinguishing advantage of $\mathcal{A}$.

Lemma 2. *Let X and Y be two probability distributions over the sample space $\mathcal{S}$, then*

$$\Delta(X, Y) = 1 - E_{s \sim Y}[\min(1, \frac{Pr[X = s]}{Pr(Y = s)})],$$

Chapter 5

Authenticated Encryption Using Cryptographic Module

Authenticated Encryption, as discussed in the previous chapter is the cryptographic primitive, which is used to provide both confidentiality and authenticity simultaneously. There are several authenticated encryption algorithms in existence already. However, these algorithms are often difficult to use correctly in practice. . This is a significant problem because misusing AE constructions can result in reduced security in many cases. Furthermore, many existing algorithms have numerous undesirable features. for example, the concept of online authenticated encryption was proposed in [38], in which encryption can be done on the fly. If the message (resp. ciphertext) blocks are denoted by $M_1, M_2, \ldots$ (resp. $C_1, C_2, \ldots$), then the requirement of online AE means that the ciphertext block C_i can be computed without the knowledge of plaintext block M_j for any $j > i$. Most of the block-cipher based authenticated encryption schemes can be used in online encryption mode.

However, decryption in an AE scheme can never be online due to the fact that the ciphertext needs to be verified before producing the plaintext. If this was not the case then an attacker could simply ask for the decryption of a ciphertext of his choice, while associating any arbitrary tag with the ciphertext.The requirement of verifying the tag before producing the decrypted text can be solved in two different ways as shown in Fig. 5.1: Decrypt-then-Verify and Verify-then- Decrypt.

In Verify-then-Decrypt, the system first applies the tag verification algorithm and then produces the plaintext, block by block, only if the tag verifies. This method can't be implemented in less than 2 passes over the ciphertext which eventually results in longer runtime. In Decrypt-then-Verify, the system implementing the decryption process could store the complete plaintext and produce it only if the tag verifies. However, for a long message this method requires a potentially large memory on the device to store the complete plaintext.

In practice, most cryptographic designs are implemented inside a cryptographic module, as suggested by National Institute of Standards and Technology (NIST) in a set of standards

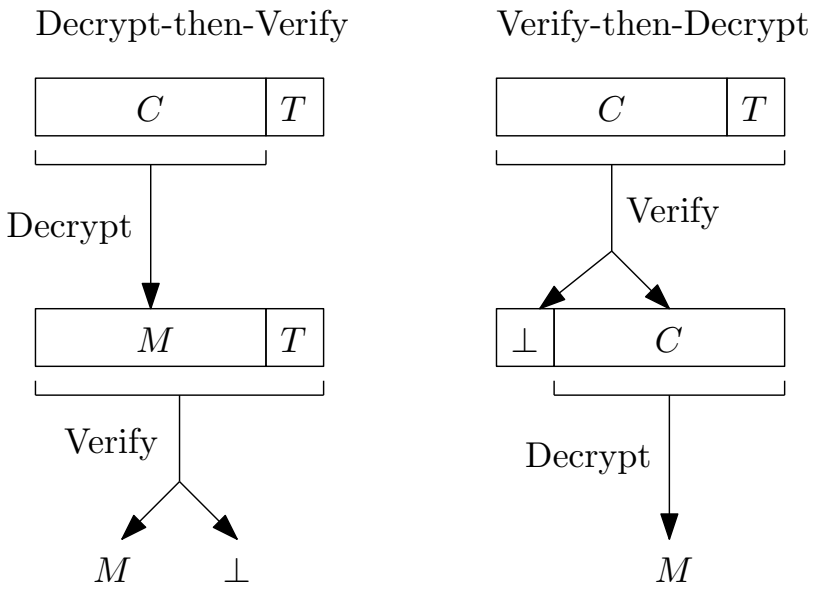


Figure 5.1: Authenticated Decryption

(FIPS 140). A cryptographic module has a limited memory. Hence, this makes it challenging to implement a authenticated encryption for long messages inside it. In a series of standards (FIPS 140), NIST already recommends the use of a cryptographic module for stream ciphers, block ciphers, authenticated encryptions etc.

5.1 Cryptographic Module

A cryptographic module is a device (plus the requisite software to make it work) with a secret key stored inside it and it is assumed that the device itself is secure against all types of adversaries. Cryptographic modules are usually secured physically and the attacker is not allowed access to the input of the device. The only part where the attacker can mount an attack is the output of the module. In general, it is hard to keep a big and bulky device from attackers and hence such a device usually contains a small amount of memory. This memory could range from few bytes for tiny devices to few kilobytes.

As mentioned above, a major limitation of a cryptographic module is the limited memory. However, as we discussed, decryption in authenticated encryption requires a large memory to store the entire plaintext inside it. Therefore, it is not feasible to implement a conventional authenticated encryption on a cryptographic module.

The Federal Information Processing Standards (FIPS) specify requirements for cryptographic modules for various security algorithms in FIPS 140 series. The standard FIPS 140-2 [1] defines a cryptographic module as: the set of hardware, software, firmware, or some combination thereof that implements cryptographic logic or processes, including cryptographic algorithms, and is contained within the cryptographic boundary of the

module. The standard [1] specifies the security requirements satisfied by a cryptographic module when implemented within a security system protecting sensitive but unclassified information. It provides four increasing, qualitative levels of security that are intended to cover the wide range of potential applications and environments in which cryptographic modules may be employed. The available cryptographic services for a cryptographic module are based on many factors that are specific to the application and the environment. The four security levels as defined in [1] are as follows.

- **Security Level 1:** It provides the lowest level of security by supporting basic security requirements specified for a cryptographic module (e.g., use of at least one approved algorithm or approved security function). This level does not require any specific physical security mechanisms.
- **Security Level 2:** This level enhances the physical security mechanisms of Level 1 by adding the requirement for tamper-evidence, which includes the use of tamper-evident coatings or seals or for pick-resistant locks on removable covers or doors of the module.
- **Security Level 3:** This level enhances the physical security mechanisms required at Level 2. It attempts to prevent the intruder from gaining access to critical security parameters held within the cryptographic module. This level is intended to provide physical security requirements with a high probability of detection and response to attempts at physical access, use or modification of the cryptographic module.
- **Security Level 4:** This level provides the highest level of security defined in the standard. The physical security mechanisms provide a complete envelope of protection around the cryptographic module. The implemented mechanisms intent of detecting and responding to all unauthorized attempts at physical access. There is a very high probability of detection of penetration of the cryptographic module from any direction and the detection results in the immediate zeroization [1] of all plaintext critical security parameters.

5.2 Operational Environment

In this section, we have explained the general setup, which we will consider in next chapters while proposing the design of new schemes. Our scenario assumes three entities, namely the sender, the receiver, and the cryptographic module, which are interacting in the protocol. We assume that the encryption and decryption functionality is carried out inside the cryptographic module, which has limited storage capability and a built-in secret key. The purpose of using the cryptographic module is to provide a secure practical setup to prevent the leakage of the secret key from the device.

The sender provides a message M to the encryption algorithm, which passes the message block by block to a cryptographic module, which do the further processing and returns the ciphertext blocks and tag (C, T)as a output to the receiver. Receiver then provide (C, T)

as a input to the decryption algorithm, which in turn it passes block by block to the cryptographic module. Cryptographic module decrypt the ciphertext without storing the decrypted blocks and compute the tag. During the decryption, it just stores the small intermediate state or some other info called S. If the tag verifies, it returns that S value to the receiver, using which and C, receiver can itself compute actual message M. If tag does not verify, then cryptographic module just returns invalid symbol $\perp$.

Chapter 6

Related Work

In last chapter, we explained the issues with authenticated encryption when implemented inside a cryptographic module. Due to the restriction of the memory within cryptographic module, it is not feasible to implement conventional AE scheme inside it. However, in practice, sometimes it becomes mandatory to implement AE inside cryptographic module. In this chapter, we have explained some of the existing solutions in this direction followed by their limitations.

6.1 Intermediate Tag

Intermediate tag is the most simplest solution for handling the above mentioned issue in case of long messages. Under this technique, instead of generating one tag for whole message, several tags are generated. For a given message M, we divide M into n blocks of b bits each where $M_1||M_2 \dots M_n \leftarrow M$. These b bits blocks are further grouped into small chunks called size s and separate authentication tag is computed for each small chunk. So, if a message consists of n blocks and size of each chunk is s i.e. each chunk consists of s blocks, then the total no of chunks can be given by $c = \lceil n/s \rceil$. An individual tag ig generated for each chunk, hence the total number of tags will be c. The following diagram explains the concept of intermediate tag.

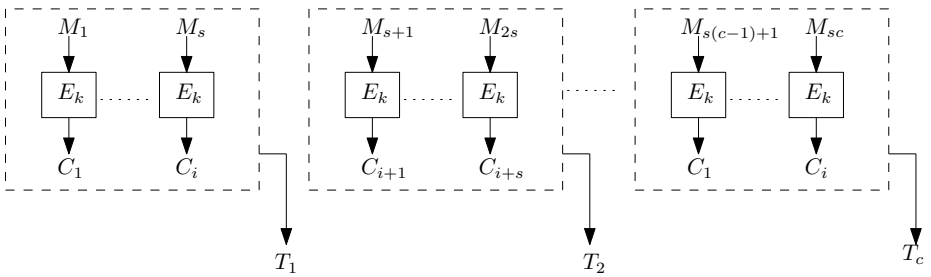


Figure 6.1: Intermediate Tag Generation

In this way when using intermediate tags, we do not need to store whole plaintext in the memory. However, just storing the small chunks that can be easily occupied in the available memory will suffice. And verification is done on each chunk, if the verification succeeds for that chunk, it is released to the user and the processing of the next chunk begins.

Limitations:

- The intermediate tag approach is not safe, if forgery is at end. In that case, user has already got the initial chunks of the message, which he can used to mount an forgery attack.
- Enough memory space needed for storing multiple tags.

6.2 Remotely Keyed Authenticated Encryption

Blaze [20] proposed a new paradigm for secret-key block ciphers: Remotely keyed encryption (RKE). RKE is concerned with the problem of "high-bandwidth encryption with low bandwidth smartcards". A scheme to achieve the same was proposed in this work, but some limitations of the scheme were also mentioned in the same work. Later, Lucks [55] provided the first formal model for RKE and chose to interpret the question as that of implementing a remotely key pseudorandom permutation (RKPRP). The work [55] was further improved, both in terms of formal modeling and the actual construction, by an influential work of Blaze et. al [21]. It was observed in this later work that the PRP's length preserving property implies that it can not be semantically secure when viewed as encryption function alone. Thus, in addition to the RKPRP which was termed "length preserving RKE", the work [21] also introduced the notion of "length increasing RKE" which was also referred to as "remote key authenticated encryption" (RKAE). While the definition given in [21] was important and the first step towards formalizing this new notion, it turns out to be quite non-standard (it involves an "arbiter" who can fool any adversary). The notion of such an arbiter looks quite artificial. Later, Dodis et al. [32] provided the formal definition of a remote key authenticated encryption. They proposed a formalized solution to the problem of remotely keyed authenticated encryption (RKAE). One round of the proposed scheme consists of seven different algorithms that run between two parties called the host and the cryptographic module: (RKG, StartAE, CardAE, FinishAE, StartAD, CardAD, FinishAD). In this scheme, the authors used the smart card as the Crypto module. The host is assumed to be powerful but insecure, and the Crypto module is assumed to have low bandwidth and limited memory but taken to be secure. For a given authenticated encryption scheme AE, we explain the RKAE using an example. First, the RKG runs and produces the secret key K and this key gets stored on the Crypto module. The process of encryption is divided into 3 parts (shown in Fig. 6.2(a)):

1. First, the host runs a probabilistic algorithm *StartAE* on input M (which we conveniently rename *Conceal* and produces (h, b) pair, where $|b| << |h|$, $h \leftarrow E_\tau(M), b = \tau \| H(h)$, E is any one time symmetric encryption algorithm, τ is the

session key and $H(.)$ is a collision resistant hash function. Next, b is sent to the Crypto module while h will be used as a part of the final ciphertext.

2. On receiving b, the Crypto module runs an authenticated encryption algorithm CardAE, which could be any AE scheme using secret key K, which produces $C = AE_K(b)$ and sends it to the host. Note that the key K is only known to the Crypto module.

3. At the end, the host runs FinishAE by producing $C' = \langle C, h \rangle$ pair as the resulting authenticated encryption of M.

Similarly, the process of decryption is divided into 3 parts (shown in Fig. 6.2(b)):

1. For decryption, the host initially has the pair $C' = \langle C \| h \rangle$, out of which he sends C to the Crypto module.

2. Crypto module receives C and runs CardAD which is an authenticated decryption algorithm AD using secret key K, and outputs $b = AD_K(C)$ that will be sent to the host.

3. Finally, the host runs FinishAD (which we conveniently rename Open(h, b)). This deterministic algorithm Open(h, b) outputs M if (h, b) is a"valid" pair, otherwise it returns the invalid operator $\perp$. Open(h, b) first parses b into $\tau \| H(h)$, and then uses τ to decrypt h.

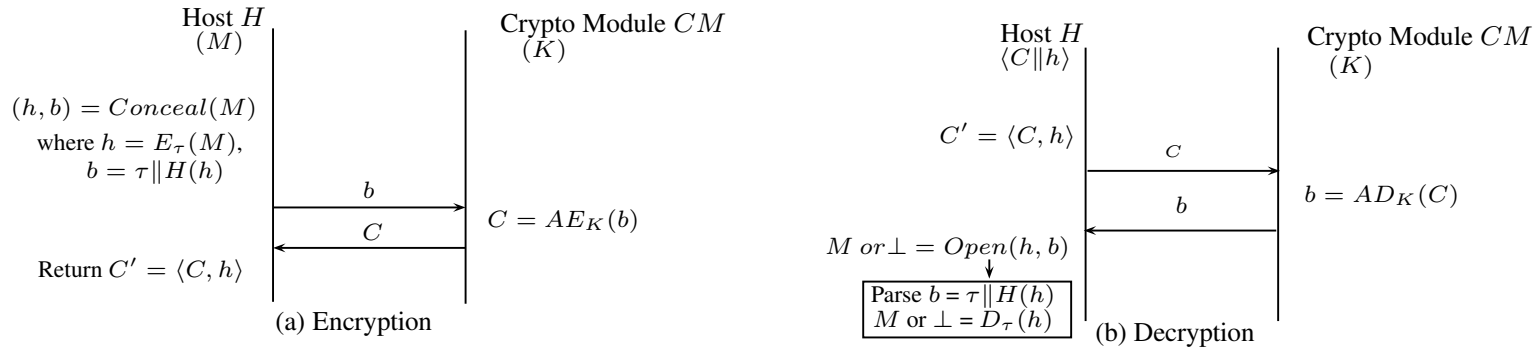


Figure 6.2: Remote Key Authenticated Encryption

Limitations

The RKAE scheme solves the problem where one wishes to split the task of high bandwidth authenticated encryption between a secure client (but limited bandwidth or computationally limited device) and an insecure (but computationally powerful) host. Though RKAE is efficient, but the Crypto device (e.g. a card) that performs encryption and

decryption does not know the actual value of plaintext and ciphertext . Rather, it trusts the insecure host to provide these values, making it unsuitable for real world applications. Further, the security of this scheme is proved in a setting where the adversary has oracle access only to the combined functionality of the host and the card. However, the host is assumed to be insecure (subject to break-in by an adversary) in this scheme, i.e., the adversary can have access to the internal state of an encryption algorithm executed on the host side. So, the underlying assumption of having oracle access to combined functionality is not justified for defining security of the scheme.

6.2.1 Decrypt-Then-Mask Protocol

In [38], Fouque et al. proposed a generic construction called Decrypt-then-mask. Their construction does not affect encryption, only the decryption part is modified. They use a pseudorandom number generator to mask the plaintext blocks, to eliminate the need of storing plaintext blocks. Once the tag gets verified, the scheme returns the seed of the PRNG that has been used to mask the plaintext blocks. We use CM to denote cryptographic module and U to represent user of this device in further description of the scheme. For a given authenticated encryption scheme AE, the encryption and decryption component work as follows (shown in Fig. 6.3):

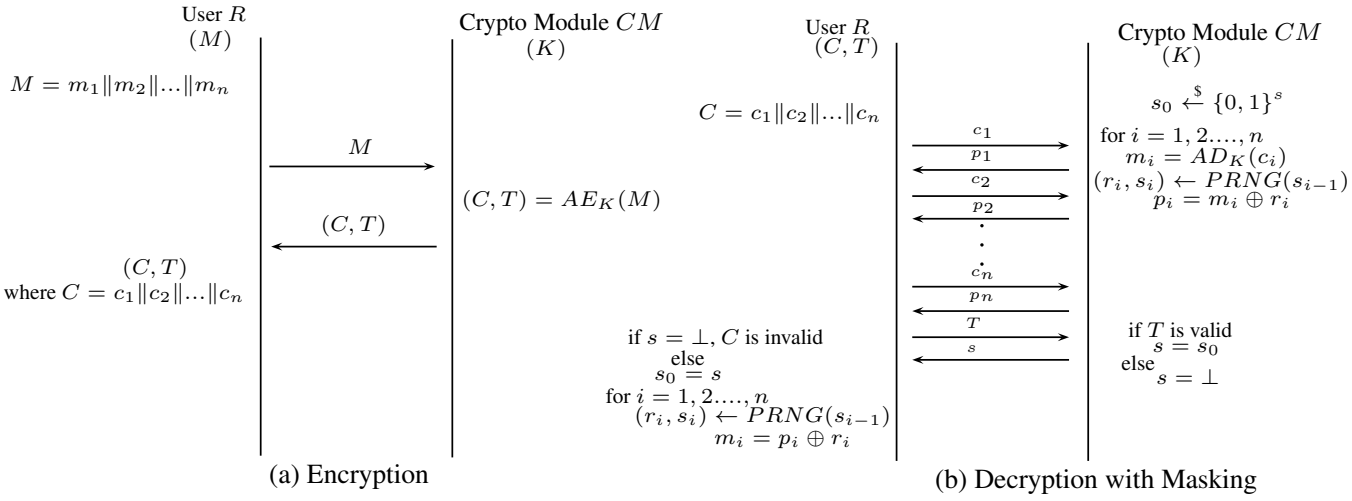


Figure 6.3: Encryption and Decryption using Decrypt-Then-Mask Protocol: In (b)decryption, to represent it diagrammatically we assume $AD_K(c_i)$ returns the intermediate value m_i.

1. User U is given ciphertext-tag (C, T) pair,where $C = c_1\|c_2\|...\|c_n$, and he wants to decrypt it only if it has not been modified.

2. Crypto module CM executes the seed generation algorithm to generate a random seed s_0 for the PRNG and generates (r_1, s_1) = PRNG(s_0). User U sends first ciphertext block c_1 to CM. Crypto module CM initializes the tag computation and decrypts c_1 to generate $m_1 = AD_K(c_1)$ and masks m_1 to obtain $p_1 = m_1 \oplus r_1$ and sends it to U. In general, $AD_K(.)$ takes (C, T) pair as input and returns plaintext M or invalid operator $\perp$ (if the tag T is not valid). For better explanation, we assume $AD_K(c_i)$ returns the intermediate m_i.

3. U sends ciphertext block one by one to CM. Upon receiving each ciphertext block c_i, it updates the tag computation and decrypt c_i followed by masking $p_i = m_i \oplus r_i$ where r_i is generated as (r_i, s_i)= PRNG(s_{i-1}). Then p_i is returned back to the user U. This process repeats until all ciphertext blocks get processed.

4. Once the processing of all ciphertext blocks get done, U sends tag T to CM and the Crypto module CM checks validity of the tag. If the tag is valid then it returns s_0 to U, otherwise it returns invalid symbol $\perp$.

5. If U receives s_0, i.e., tag is valid and it decrypts $p_1 \| p_2 \| .... \| p_n$. For $i = 1, 2...n$, it generates (r_i, s_i)= PRNG(s_{i-1}), and decrypt p_i as $m_i = p_i \oplus r_i$. Finally plaintext $M = m_1 \| m_2 \| .... \| m_n$. If U receives $\perp$ then the tag is invalid and the plaintext M can not be generated.

The main idea behind this protocol is to blind the plaintext blocks obtained after decryption by XORing them with a pseudorandom sequence of bits and return it to the sender. These blocks do not look meaningful to the attacker until unmasking is applied. Once a tag gets verified at the receiver end, the seed of the pseudorandom number generator(PRNG) used to mask the plaintext blocks is returned to the sender. This allows the sender to run the same PRNG and un-mask the plaintext blocks.

Limitations

- The DTM requires two additional passes excluding no. of passes required for underlying AE during decryption, which makes it computationally expensive, specially in the case of long messages. These two extra passes (one at the crypto module and another one at the user side) in DTM are due to the use of PRNG, which creates extra overhead particularly in case of long message.

6.3 Releasing Unverified Plaintext

Andreeva et. al addressed the issue of releasing unverified plaintext and formalized it as the RUP settings [35]. Under these RUP settings, plaintext or the information relating to it is released to the attacker prior to the verification, eliminating the requirement of memory to store the entire plaintext. However, it does not mean omitting verification, which remains essential to preventing incorrect plaintexts from being accepted. In [35], they provided 2 new security notions called PA (Plaintext Awareness) and INT-RUP (Integrity under

Releasing Unverified Plaintext). To provide privacy PA is used along with IND-CPA. At the heart of PA notion is plaintext extractor. An encryption scheme is said to be PA if there exists an efficient plaintext extractor for every adversary that mimics the decryption oracle in order to fool an adversary. Plaintext extractor is not allowed to make the encryption nor decryption queries and does not even have information about the secret key. There are two notions of plaintext awareness: PA1 and PA2. In PA1 settings, extractor is given access to the history of queries made to encryption oracle, but not in PA2. Hence, PA2 is more stronger notion than PA1. An AE scheme is INT-RUP if adversary can not find a new ciphertext tag pair which passes the verification, with having access to the encryption and unverified decryption oracle.

Limitations:

- State-of-the-art research give the impression that achieving RUP security by insignificantly altering existing schemes is out of reach: all designs providing such security either require significant changes, a completely new design, or an extra pass, making the schemes slower and adding design complexity.

Chapter 7

sp-AELM: Sponge based Authenticated Encryption for Limited Memory Devices

A cryptographic module should not reveal any information about the plaintext until ciphertext-tag pair gets verified. Once the ciphertext gets verified, then crypto device is allowed to reveal the plaintext. However, a cryptographic module is likely to have some storage restriction. Therefore, it can not store a large plaintext M, verify the tag and output M only when M is valid. Traditional authenticated encryption schemes do not satisfy this usage scenario.

In this chapter, we introduce a new generalized technique for authenticated encryption which store only one intermediate state instead of entire plaintext during tag verification. And then this intermediate state is used by the user on their side to decrypt ciphertext. Without releasing unverified plaintext this technique allow us to do a tag verification for a long message on the devices with limited memory settings. We explain this generalized technique using our new construction sp-AELM and its variants. This new construction and its two more variants support both online encryption as well as tag verification in constrained memory setting for long messages, which makes it suitable for implementation on devices that have limited memory. The Sponge construction supports arbitrarily long input and output sizes, and hence allows building various cryptographic primitives such as a hash function or a stream cipher [15]. More recently, use of Sponge based hash functions as an authenticated encryption primitive has been proposed in [14]. Keccak, the winner of the SHA-3 competition, is also built on the Sponge construction. Considering the acceptability and future adaptability of the Sponge construction in Cryptographic software/hardware, we choose the Sponge function as a basic primitive for the construction of our authenticated encryption scheme sp-ALEM.

In sp-AELM, instead of returning all the plaintext blocks, we provide only one internal state to the user, using which plaintext can be computed easily at his side. Our scheme requires a total of three passes, one for encryption and two for decryption. This compares

favorably with the DTM scheme, which requires overall more number of passes (discussed in Table 7.1), making our scheme more efficient. In sp-AELM, all computations are propose to be done inside a cryptographic module, where the actual plaintext and ciphertext values are securely known, as opposed to the RKAE scheme [32]. More details about sp-AELM are given in Section 7.2. In addition to this, we also present two more efficient variants of sp-AELM. More detail about these variants are given in Section 7.5.

7.1 Sponge Function

A sponge has the same interface as a random oracle: it takes an input bitstream of any length and produces an output stream of any desired length based on a fixed-length permutation f operating on b bit state. Here, state of b bit is divided into two parts:r and c bits, where r is called rate and c is the capacity. Its structure is shown in Fig. 7.1.

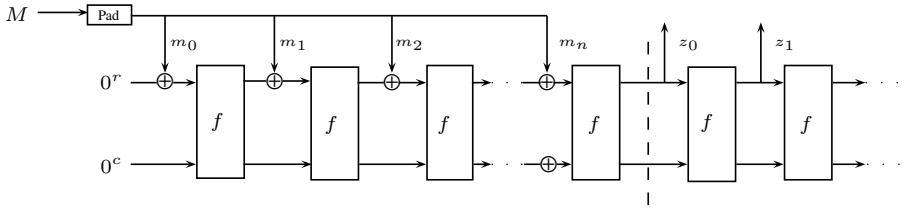


Figure 7.1: Sponge Function

First, input message is padded and divided into blocks of r bit, and state (b bit) is initialized to zero. Then it proceeds in two phase:

1. Absorbing Phase : In this phase r bit input block is XORed with first r bits of state and then f is applied to whole b bits. Once all input blocks are absorbed, it moves to the squeezing phase.

2. Squeezing Phase: In this phase first r bits of state returned as output followed by applying f. Number of output block depends on user interest.

7.2 Description of sp-AELM

We now describe the new authenticated encryption scheme sp-AELM that addresses the low memory issue mentioned earlier. Schematic structure of sp-AELM is shown in Fig. 7.2. Our authenticated encryption scheme takes key K, nonce N, associated data A, plaintext M and $flag$ as inputs and returns C or (C, T) depending on the value of the $flag$, where C represents the ciphertext and T denotes tag or message authentication code (MAC). We denote cryptographic module by CM and the user of this module by U. The user U uses the crypto module CM for encryption, decryption and verification. The encryption and decryption procedures are given in Algorithm 3 and Algorithm 4 respectively.

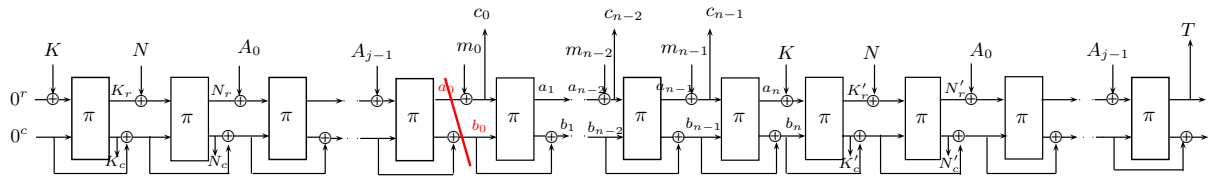


Figure 7.2: sp-AELM Construction

7.2.1 Encryption

The algorithm for encryption, $\mathcal{E}_K(.,.,.,.)$ is explained in Algorithm 3. It takes an input, the secret key K of r bits, nonce N of r bits and associated data A, plaintext M and a $flag$ value. This flag value can be 0 or 1, where 0 represents continue i.e. partial M is provided as an input and 1 represents stop i.e., M is complete, we can do the tag computation. Associated data A is partitioned into r bit blocks$(A_0\| \dots A_{j-1})$ and 10^* padding is used for last block if required. We use two initialization vectors IV_1 and IV_2 of size r and c respectively, each of which is initialized to zero. Further, flag value is checked, if it is 1, then the message M is padded using $Pad(M)$ and divided into n blocks $(M = m_0\|m_1\| \dots m_{n-1})$, each block is of r bit (i.e. $|M|/r = n$). Otherwise it is just processed into r bit blocks. Algorithm iterates a fixed permutation $\pi : A \times B \to A \times B$, where $|A| = r$ *and* $|B| = c$. First $(IV_1 \oplus K)\|IV_2$ is given input to π and the output is $K_r\|K_c$. Then IV_2 is XORed with K_c(say this value X). The next input to π is $(K_r \oplus N)\|X$ and the corresponding output is $N_r\|N_c$. Then this N_c values is XORed with X. Similarly associated data A is processed. After that message block $m_i's$ for $i = 0, 2, ..(n-1)$ get processed. Note that encryption is online here, we do not need to know message blocks in advance i.e., c_i can be calculated without prior knowledge of m_j for any $j > i$. Once all the message blocks are processed and $flag$ is 0, then $C = c_0\|c_1\|....\|c_{n-1}$ is return to the user and algorithm terminates. Otherwise K, N, A are processed again in a order followed by tag generation. Tag T is generated from Z_0 value. Z_0 is the initial r bits of final output of π and tag T is extraction of required initial bits of Z_0 (if tag is of 128 bit, then T is initial 128 bit of Z_0). Finally (C, T) pair is returned to the user, where $C = c_0\|c_1\|....\|c_{n-1}$.

Algorithm 1: Encryption $\mathcal{E}_K(N, A, M, flag)$

Initialization: $IV_1 = 0^r, IV_2 = 0^c$,
  $K_r \| K_c \xleftarrow{\$} \{0,1\}^b$,
  $I_\pi = \{((IV_1 \oplus K) \| IV_2, K_r \| K_c)\}$
if $(flag = 1)$ **then**
  $M = m_0 \| m_1 \| ..... \| m_{n-1}$, Where $|m_i| = r$ and $0 \leq i < (n-1)$
  $Pad(M) =$ $m_0 \| m_1 \| ..... \| (m_{n-1} || 10^{r-(|m_{n-1}|+1)})$
else
  $M = m_0 \| m_1 \| ... \| m_{n-1}$, Where $|m_i| = r$
  if $|m_{n-1}| < r$ **then** {return Invalid;}
$A = A_0 \| A_1 \ldots \| A_{j-1}$ s.t. $|A_i| = r$ and $0 \leq i < (j-1)$
$Pad(A) =$ $A_0 \| A_1 \ldots \| (A_{j-1} || 10^{r-(|A_{j-1}|+1)})$
$x = K_r \oplus N$, $w = K_c$
$N_r \| N_c = \pi(x \| w)$
$x = N_r$, $w = N_c \oplus w$
for $i = 0 \rightarrow j-1$ **do**
  $x' \| w' = \pi(x \oplus A_i \| w)$
  $w = w' \oplus w, x = x'$
$a_0 = x', b_0 = w'$
$c_0 = a_0 \oplus m_0$
for $i = 1 \rightarrow n-1$ **do**
  $x^{'} \| w^{'} = \pi(c_{i-1} \| b_{i-1})$
  $b_i = b_{i-1} \oplus w^{'}$
  $a_i = x^{'}$
  $c_i = a_i \oplus m_i$
$C = c_0 \| c_1 \| ..... \| c_{n-1}$
if $(flag = 0)$ **then**
  Return C; Exit
$a_n \| b_n = \pi(c_{n-1} \| b_{n-1})$
$x = a_n, w = b_n$
$x = x \oplus K$
$K_r^{'} \| K_c^{'} = \pi(x \| w)$
$x = K_r^{'} \oplus N$, $w = K_c^{'} \oplus w$
$N_r^{'} \| N_c^{'} = \pi(x \| w)$
$x = N_r^{'}$, $w = N_c^{'} \oplus w$
for $i = 0 \rightarrow j-1$ **do**
  $x' \| w' = \pi(x \oplus A_i \| w)$
  $w = w' \oplus w, x = x'$
$T = Z_0 = x$
Return (C, T)

Algorithm 2: Decryption $\mathcal{D}_K$ (N, A, C, T)

Initialization: $IV_1 = 0^r, IV_2 = 0^c$,
  $K_r \| K_c \xleftarrow{\$} \{0,1\}^b$,
  $I_\pi = \{((IV_1 \oplus K) \| IV_2, K_r \| K_c)\}$
$A = A_0 \| A_1 \ldots \| A_{j-1}$ s.t. $|A_i| = r$ and $0 \leq i < (j-1)$
$Pad(A) =$ $A_0 \| A_1 \ldots \| (A_{j-1} || 10^{r-(|A_{j-1}|+1)})$
$x = K_r \oplus N$, $w = K_c$
$N_r \| N_c = \pi(x \| w)$
$x = N_r$, $w = N_c \oplus w$
for $i = 0 \rightarrow j-1$ **do**
  $x' \| w' = \pi(x \oplus A_i \| w)$
  $w = w' \oplus w, x = x'$
$a_0 = x, b_0 = w$
$m_0 = a_0 \oplus c_0$
for $i = 1 \rightarrow n-1$ **do**
  $x^{'} \| w^{'} = \pi(c_{i-1} \| b_{i-1})$
  $b_i = b_{i-1} \oplus w^{'}$
  $a_i = x^{'}$
  $m_i = a_i \oplus c_i$
$M = m_0 \| m_1 \| ..... \| m_{n-1}$
$x = a_n, w = b_n$
$x = x \oplus K$
$K_r^{'} \| K_c^{'} = \pi(x \| w)$
$x = K_r^{'} \oplus N$, $w = K_c^{'} \oplus w$
$N_r^{'} \| N_c^{'} = \pi(x \| w)$
$x = N_r^{'}$, $w = N_c^{'} \oplus w$
for $i = 0 \rightarrow j-1$ **do**
  $x' \| w' = \pi(x \oplus A_i \| w)$
  $w = w' \oplus w, x = x'$
$Z_0 = x$
if $(Z_0 == T)$ **then**
  Return (a_0, b_0)
else
  Return $\perp$

7.2.2 Decryption and Verification

The algorithm for decryption $\mathcal{D}_K(.,.,.,.)$ is given in Algorithm 4. The decryption process is similar to encryption. All the steps are same except for XORing with the message. The only difference is rather than XORing with m_i, we XOR with c_i. Once the message block m_i is evaluated we use that as input just like encryption. Here in this decryption algorithm, we just store (a_0, b_0) value, all $m_i's$ block not get stored anywhere. Finally tag Z_0 is calculated. Once the tag becomes available , the given tag T is compared with Z_0. If tag gets verified, i.e., $Z_0 = T$, then algorithm returns (a_0, b_0) value. Once the user get this value, he can easily recover plaintext M from C as $m_0 = c_0 \oplus a_0$, $m_1 = c_1 \oplus \pi(c_0, b_0)$[initial r bit] and so on (explained in Algorithm 4). If tag does not get verified, then it returns invalid operator, i.e., plaintext cannot be calculated in this case. Fig. 7.3(b) shows the decryption protocol used in this new scheme.

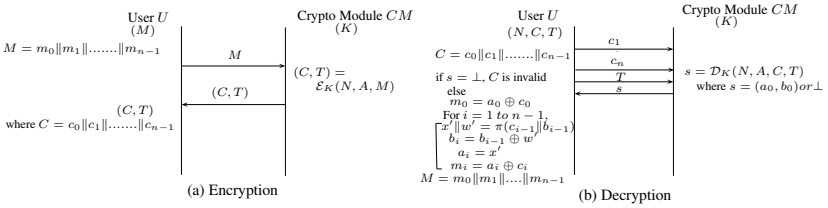


Figure 7.3: Encryption and Decryption protocol for sp-AELM

7.3 Definitions

Definition 8. ***(Ideal Online Function)*** *Let* ***func****(A, B) be defined as set of all functions from set A to set B. Now, we define ideal online function $\$_{pad}$ as:*

$$\$_{pad}(N, A, M, flag) \rightarrow \begin{cases} (C,T) & \text{if } flag \text{ is } 1, \\ C & \text{if } flag \text{ is } 0, \end{cases}$$

where,

$pad(X) = X\|10^t$ where t is a non-negative integer s.t $|X\|10^t| = pr$ for $p > 0$ and some fixed r.

N= nonce of r bit,

A= Associated data, where $|A| < r$ and A'=pad(A),

M= input message that can be partial or complete depending on the flag value,

$$flag = \begin{cases} 0 & ; \text{ if user has queried partial message } M\text{ ,} \\ 1 & ; \text{ if user has queried complete message } M\text{ .} \end{cases}$$

In case $flag = 0$, we are assuming user has supplied incomplete message and wlog length of supplied message ($|M|$) is multiple of r.

$$M' = \begin{cases} m_0\|\ldots\|m_{n-1} = M, where\ n = |M|/r\ and\ |m_i| = r & ;\ if\ flag{=}0 \\ m_0\|\ldots\|m_{n-1} = pad(M)\ s.t.\ |m_i| = r\ for\ 0 \le i \le n-1 & ;\ if\ flag{=}1\ . \end{cases}$$

$C = c_0\|c_1\|\ldots\|c_{n-1}$ *where* $c_j = g(N, A', m_0\|\ldots\|m_j)$ *for* $j = 0, \ldots, n-1$, *and*
$g \xleftarrow{\$} func(\{0,1\}^r \times \{0,1\}^r \times (\{0,1\}^r)^+, \{0,1\}^r)$,
$T = g'(N, A', M')$, *where* $g' \xleftarrow{\$} func(\{0,1\}^r \times \{0,1\}^r \times (\{0,1\}^r)^+, \{0,1\}^t)$.

Notice that above function is online: here prefixes of outputs remain the same if prefixes of the inputs remain constant. Furthermore, if we consider that nonce and associated data are unique for each function invocation in the ideal online function, then we acheive full privacy. This is because the inputs to g and g' will then be unique for each invocations, and since g and g' are functions randomly chosen from $func$, we get outputs that are independent and uniformly distributed.

Definition 9. ***(Privacy)*** *For a given authenticated encryption scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, we define privacy in terms of the ability of an adversary A to distinguish between the output of an encryption oracle $\mathcal{E}$ from the output of an ideal online function $\$_{pad}$. We define*

$$Adv^{priv}_{\mathcal{AE}}(A) = Pr[K \xleftarrow{\$} \mathcal{K} : A^{\mathcal{E}_K(.,.,.,.)} \Rightarrow 1] - Pr[A^{\$_{pad}(.,.,.,.)} \Rightarrow 1]$$

The oracle $\mathcal{E}_K$ on input $(N, A, M, flag)$, returns the C or (C, T) depending on the flag value whereas the oracle $\$_{pad}$ on input $(N, A, M, flag)$, works as defined in Def. 1. Here the advantage of an adversary A will be its ability to distinguish between the ciphertext outputs from the $\mathcal{E}_K$ and $\$_{pad}$.

Definition 10. ***(Authenticity)***. *Here we give a notion of authenticity of the ciphertext tag pair of an authenticated encryption scheme. For a given authenticted encryption scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, let A be an adversary having an encryption oracle $\mathcal{E}_K$. We say that A forges if it outputs a (N, A, C, T) tuple where $\mathcal{D}_K(N, A, C, T) \neq INVALID$ and adversary A didn't ask a query $\mathcal{E}_K(N, A, M, flag)$ that resulted in response (C, T).*
Let $Exp^{Auth}_{\Pi}(A)$ be the forging experiment for a given $\mathcal{AE}$. Then the forging experiment is defined as follows:

1. *Adversary A can query encryption oracle $\mathcal{E}$, decryption oracle $\mathcal{D}$ atmost q_{enc}, q_{dec} times respectively.*

2. *All the query responses of encryption oracle $\mathcal{E}$, are stored in a set, say R. This R set contain (N, A, C, T) tuple.*

3. *If adversary A is able to generate a (N, A, C, T) tuple $\notin R$ that produces valid message M, then he wins and output is 1 otherwise output is 0, after trying q_{dec} number of queries.*

We say adversary wins i.e. he succeed in creating forgery if $Exp_{AE}^{Auth}(A) = 1$. *So, the advantage of adversary in forging the scheme is defined as:*

$$Adv_{AE}^{Auth}(A) = Pr[Exp_{AE}^{Auth}(A) = 1]$$

7.4 Security Proof

In this section, we provide security proofs for the privacy and authenticity of spALEM. We have used recently evolved game playing technique proposed by Bellare and Rogaway in [12]. We prove security of sp-AELM in the ideal permutation model, where the underlying permutation is assumed to behave perfectly random. We also restrict our proof to the assumption that nonce N will always be generated different and randomly by the attacker. As it is not practically feasible by the algorithm to generate it randomly and differently everytime. To maintain the simplicity of proof, we have considered associated data of one block only. This can be further extended with minor changes in proof.

7.4.1 Privacy

We obtain an upper bound for the advantage of the adversary who can distinguish the output of the proposed scheme with a random oracle in the ideal permutation model.

Theorem 1. *Let* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *denote the proposed Authenticated Encryption scheme with defined padding rule and an permutation* π, *which operates on* b *bits. The adversary* A *has given access to* π, π^{-1}. *Then the advantage of* A *relative to* $\mathcal{E}$ *is given by*

$$Adv_{\Pi}^{priv}(A) = Pr[K \xleftarrow{\$} \mathcal{K} : A^{\mathcal{E}_K(),\pi,\pi^{-1}} = 1] - Pr[K \xleftarrow{\$} \mathcal{K} : A^{\$_{pad}(),\pi,\pi^{-1}} = 1] \ .$$

$$Adv_{\Pi}^{priv}(A) \leq \frac{\sigma(\sigma-1)}{2^b+1} + \frac{\sigma(\sigma-1)}{2^c+1} + \frac{(q_\pi + q_{\pi^{-1}}).q_{enc}}{2^r} + \frac{2(q_\pi + q_{\pi^{-1}})\sigma}{2^c - \sigma} + \frac{q_\pi}{2^r} \ .$$

where $\$_{pad}$ *is defined in Def. 1,* σ *is the maximum number of block calls to* π, π^{-1} *by encryption* $\mathcal{E}$ *and decryption* $\mathcal{D}$ *algorithm .* q_{enc}, q_π *and* $q_{\pi^{-1}}$ *are the maximum number of queries to encryption oracle Enc,* π *and* π^{-1} *oracle respectively by adversary* A. $b(= r + c)$ *is the size on which* π *permutation operates.*

Proof. The advantage of the adversary is his ability to distinguish the proposed scheme from a random oracle. We used the game based framework to compute this advantage. We define a sequence of nine games from $G0$ to $G8$ given at the end of this chapter, where Game $G0$ represents the proposed scheme and $G8$ represents completely random output. Computations for probability difference between consecutive games are provided below.
Game G0: In this game encryption oracle perfectly simulates the proposed algorithm $AE - E^{\pi,pad}$ and π, π^{-1} oracle simulate an ideal permutation and its inverse.

$$Pr[A^{E_k,\pi,\pi^{-1}} = 1] = Pr[A^{G0} = 1]$$

Game G1: Game $G1$ is exactly same as game $G0$.

$$Pr[A^{G0} = 1] = Pr[A^{G1} = 1]$$

Game G2: Game G1 and G2 differs only when bad event occurs, otherwise, in absense of bad event, both games are same.

$$| Pr[A^{G1} = 1] - Pr[A^{G2} = 1] | \leq Pr[bad]$$

Bad event occurs when input to π and π^{-1} collide with the elements in set I_π. So, the probability of bad event will be equal to the probability of collision in π and π^{-1}. Let $Pr[coll]$ be the probability of collision in π and π^{-1}, then

$$Pr[bad] \leq Pr[coll]$$

Let $Pr[coll_i] = (i-1)/2^b$ be the probability that there is no collision till $i-1$ queries and it occurs in i^{th} query. Thus $Pr[coll]$ can be calculated as:

$$\begin{aligned} Pr[coll] &= Pr[coll_1 \vee coll_2 \vee .....coll_\sigma] \\ &\leq Pr[coll_1] + Pr[coll_2] + ... + Pr[coll_\sigma] \\ &\leq \frac{1}{2^b} + \frac{2}{2^b} + ... + \frac{\sigma - 1}{2^b} \\ &\leq \frac{\sigma(\sigma-1)}{2^{b+1}}. \end{aligned}$$

Hence, the probability difference between $G1$ and $G2$ is

$$| Pr[A^{G1} = 1] - Pr[A^{G2} = 1] | \leq \frac{\sigma(\sigma-1)}{2^{b+1}}$$

Game G3: Game $G2$ and $G3$ are identical. In game $G3$, Enc oracle simulates behaviour of π, so it becomes independent of π. Hence, Game $G2$ and $G3$ are same from adversary point of view.

$$Pr[A^{G2} = 1] = Pr[A^{G3} = 1]$$

Game G4: Game $G3$ and $G4$ differ only when *bad* event occurs, otherwise both are exactly same. This *bad* event occurs if collision happens over c bits inside I_c set in $G4$.

$$| Pr[A^{G3} = 1] - Pr[A^{G4} = 1] | \leq Pr[bad]$$

And,
$$Pr[bad] \leq Pr[collision\ over\ c\ bits]$$

$$\begin{aligned} Pr[collision\ over\ c\ bits] &\leq \frac{1}{2^c} + \frac{2}{2^c} + ... + \frac{\sigma - 1}{2^c} \\ &\leq \frac{\sigma(\sigma-1)}{2^{c+1}} \end{aligned}$$

Hence, $$| Pr[A^{G3} = 1] - Pr[A^{G4} = 1] | \leq \frac{\sigma(\sigma - 1)}{2^{c+1}}$$

Game G5: Game $G4$ and $G5$ are exactly same from adversarial point of view. We have created two new sets $I_\pi^{'}$ and $I_\pi^{''}$, where, $I_\pi^{'}$ keeps store for all the queries to encryption oracle and $I_\pi^{''}$ keeps store for all the queries to π and π^{-1} oracle. Now, set I_π is union of $I_\pi^{'}$ and $I_\pi^{''}$. This does not make any difference in functionality of game $G4$ and $G5$.

$$Pr[A^{G4} = 1] = Pr[A^{G5} = 1]$$

Game G6: Game $G5$ and $G6$ are same, except $G6$ makes an additional check in sets $I_\pi^{'}$ and $I_\pi^{''}$. But from adversary's perspective both games are exactly same.

$$Pr[A^{G5} = 1] = Pr[A^{G6} = 1]$$

Game G7:Game $G6$ and $G7$ will differ in presence of bad_1, bad_2 and bad event.

$$| Pr[A^{G6} = 1]\text{-}Pr[A^{G7} = 1] | \leq Pr[bad_1] + Pr[bad_2] + Pr[bad]$$

We are assuming here that q_π and $q_{\pi^{-1}}$ query already has been queried to π and π^{-1} oracle respectively. And maximum number of encryption queries are q_{enc}.
Calculation for bad_1 event : bad_1 is an event of having collision at the first block over r bits in set $I_\pi^{''}$ for $G6$, as c bits are fixed here. So,

$$Pr[bad_1] \leq Pr[collision\ over\ r\ bits]$$

And, $$Pr[collision\ over\ r\ bits] \leq \frac{(q_\pi + q_{\pi^{-1}}).q_{enc}}{2^r}$$

Calculation for bad_2 event : bad_2 is an event of having collision over c bits in set $I_\pi^{''}$ for $G6$, as r bits can be controlled by adversary (by changing message). And these c bits are generated randomly and different each time.

$Pr[bad_2] \leq Pr[collision\ over\ c\ bits]$

And, $Pr[collision\ over\ c\ bits] \leq \frac{(q_\pi + q_{\pi^{-1}})\sigma}{2^c - \sigma}$

Calculation for probability of bad event : Here, bad is an event of having collision in the set $I_\pi^{''}$. This can happen either if collision occurs over c bit for a message M, as r bits can be controlled by an adversary or he is able to guess the key correctly. Hence,

$Pr[bad] \leq Pr[correct\ key\ guess] + Pr[collision\ over\ c\ bits]$

Here, $Pr[collision\ over\ c\ bits] \leq \frac{(q_\pi + q_{\pi^{-1}})\sigma}{2^c - \sigma}$ (same as computed above)

And, $Pr[correct\ key\ guess] \leq \frac{q_\pi}{2^r}$

So, $| Pr[A^{G6} = 1]\text{-}Pr[A^{G7} = 1] | \leq \frac{(q_\pi + q_{\pi^{-1}}).q_{enc}}{2^r} + \frac{2(q_\pi + q_{\pi^{-1}})\sigma}{2^c - \sigma} + \frac{q_\pi}{2^r}$

Game G8: In game $G7$ each ciphertext block is generated randomly, then concatenated and return to an adversary. Similarly Tag is also generated randomly. However, in Game $G8$, ciphertext and tag of desired length is selected as a random string and returned to an adversary. This move from $G7$ to $G8$ does not make any difference, as in both the games output is generated as completely random value. So,

$$Pr[A^{G7} = 1] = Pr[A^{G8} = 1]$$

Finally, using the fundamental lemma of game based framework,

$$\begin{aligned}
Adv_{\Pi}^{priv}(A) &= Pr[K \xleftarrow{\$} \mathcal{K} : A^{\mathcal{E}_K,\pi,\pi^{-1}} = 1] - Pr[K \xleftarrow{\$} \mathcal{K} : A^{\$_{pad}(),\pi,\pi^{-1}} = 1] \\
&= |\Pr[A^{G0} = 1] - Pr[A^{G8} = 1]| \\
&= |\,(Pr[A^{G0} = 1] - Pr[A^{G1} = 1])\,| \\
&\quad + |\,(Pr[A^{G1} = 1] - Pr[A^{G2} = 1])\,| \\
&\quad + |\,(Pr[A^{G2} = 1] - Pr[A^{G3} = 1])\,| \\
&\quad + |\,(Pr[A^{G3} = 1] - Pr[A^{G4} = 1])\,| \\
&\quad + |\,(Pr[A^{G4} = 1] - Pr[A^{G5} = 1])\,| \\
&\quad + |\,(Pr[A^{G5} = 1] - Pr[A^{G6} = 1])\,| \\
&\quad + |\,(Pr[A^{G6} = 1] - Pr[A^{G7} = 1])\,| \\
&\quad + |\,(Pr[A^{G7} = 1] - Pr[A^{G8} = 1])\,| \\
&\leq 0 + \frac{\sigma(\sigma-1)}{2^b+1} + 0 + \frac{\sigma(\sigma-1)}{2^c+1} + 0 + 0 \\
&\quad + \frac{(q_\pi + q_{\pi^{-1}}).q_{enc}}{2^r} + \frac{2(q_\pi + q_{\pi^{-1}})\sigma}{2^c - \sigma} + \frac{q_\pi}{2^r}\ .
\end{aligned}$$

7.4.2 Authenticity

In this section, we analyze the security of authenticity of the tag produced in our scheme. The forgery of an AE scheme is defined as the ability of an adversary A to generate a valid (N, A, C, T) tuple, without directly querying it to the encryption oracle. The adversary is allowed to make limited number of queries to encryption, decryption, π and π^{-1} oracles. For an AE scheme, we say the adversary A is successful in forging if it outputs a (N, A, C, T) tuple where $\mathcal{D}_K(N, A, C, T) \neq INVALID$ and adversary A didn't ask a query $\mathcal{E}_K(N, A, M, flag)$ that resulted in response (C, T) .

Theorem 2. *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the proposed authenticated encryption scheme with defined padding rule (pad) and ideal permutation (π) which operate on $b(= r + c)$ bits. The adversary A is given access to Encryption oracle $\mathcal{E}$, Decryption oracle $\mathcal{D}$, π and π^{-1} oracle. Then $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is forgeable with the probability*

$$Pr[Exp_{\Pi}^{Auth}(A) = 1] \leq \frac{\sigma(\sigma-1)}{2^{b+1}} + \frac{\sigma(\sigma-1)}{2^c+1} + \frac{q_{dec}}{2^{|T|}} + \frac{q_\pi}{2^r}\ .$$

where σ is maximum number of blocks to π, π^{-1} by encryption oracle $\mathcal{E}$, and decryption oracle $\mathcal{D}$, q_{dec} is the number of queries to decryption oracle, q_π is the number of queries to π oracle and $|T|$ is the tag length.

Proof: We used code base gaming framework to calculate the adversary advantage. Here we used the set of five games$(G0', G1', G2', G3', G4')$ that is same as initial five games

for privacy proof except that decryption oracle will also work here. Games are given at the end of this chapter. Our goal is to bound the adversary's advantage which is defined as follows:

$$Adv_{\Pi}^{Auth}(A) \quad = \quad Pr[Exp_{\Pi}^{Auth}(A) = 1] \tag{7.1}$$

Here, probability calculations will be same as that of initial five games of privacy explained in Section 5.1. So, we will directly use the results from there.
Calculations for $Pr[Exp_{G4'}^{Auth}(A = 1)]$: In this game $G4'$, adversary gets a success in generating valid tag pair either by choosing tag randomly after trying q_{dec} number of queries or he guesses the key correctly. Hence adversary's ability to win in this game will be:

$$\begin{aligned} Pr[Exp_{G4}^{Auth}(A = 1)] \quad &\leq \quad Pr[correct\ key\ guess] + \\ &\qquad Pr[random\ tag\ generation] \\ &\leq \quad \frac{q_\pi}{2^r} + \frac{q_{dec}}{2^{|T|}} \end{aligned}$$

On combining the results obtained in all the games, equation (1) becomes,

$$\begin{aligned} Adv_{\Pi}^{Auth}(A) \quad &= \quad Pr[Exp_{\Pi}^{Auth}(A) = 1] \\ &\leq \quad \mid Pr[Exp_{G1'}^{Auth}(A) = 1] - Pr[Exp_{G2'}^{Auth}(A) = 1] \mid \\ &\qquad + Pr[Exp_{G2'}^{Auth}(A) = 1] \\ &\leq \quad \frac{\sigma(\sigma-1)}{2^{b+1}} + Pr[Exp_{G3'}^{Auth}(A) = 1] \\ &\leq \quad \frac{\sigma(\sigma-1)}{2^{b+1}} + \mid Pr[Exp_{G3'}^{Auth}(A) = 1] - Pr[Exp_{G4'}^{Auth}(A) = 1] \mid \\ &\qquad + Pr[Exp_{G4'}^{Auth}(A) = 1] \\ &\leq \quad \frac{\sigma(\sigma-1)}{2^{b+1}} + \frac{\sigma(\sigma-1)}{2^{c+1}} + \frac{q_\pi}{(2^c - q_\pi)^2} + Pr[Exp_{G4'}^{Auth}(A) = 1] \\ &\leq \quad \frac{\sigma(\sigma-1)}{2^{b+1}} + \frac{\sigma(\sigma-1)}{2^{c+1}} + \frac{q_{dec}}{2^{|T|}} + \frac{q_\pi}{2^r} \ . \end{aligned}$$

7.5 More variants of sp-AELM

This section presents two more variants of sp-AELM. The advantage of these variants over actual construction is that it does not require feed forward operation. Schematically construction of sp-AELM variants are shown in Fig. 7.4 and Fig. 7.5. These variants support the low memory constraint in a same way as in sp-AELM by storing only one intermediate state(shown using red line in Fig. 7.4) instead of storing complete text. Security proof for these constructions are not provided here due to space restrictions, although, it can be proved in same manner as sp-AELM. Intuitively, we can say these are secure as releasing intermediate state will not result for the adversary to gain any information about key. One can choose these variants over sp-AELM, when there is very limited amount of memory, which is not even capable of storing states required for feed-forward operation.

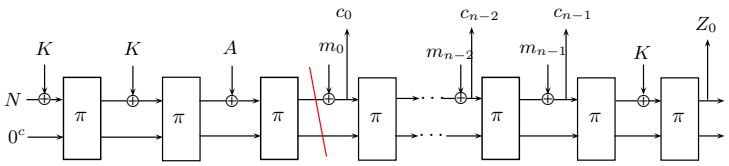


Figure 7.4: sp-AELM variant 1

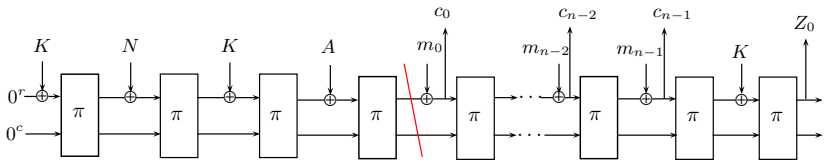


Figure 7.5: sp-AELM variant 2

7.6 Comparison

The following table shows the comparison of RKAE, DTM, the proposed scheme sp-AELM and sp-AELM variants. Our proposed scheme and its variants support online encryption. However, in the case of RKAE and DTM, it depends on the underlying *AE* scheme that is used. All these three schemes support low memory verification, i.e., there is no need to store message blocks on a cryptographic module during decryption. Next we compare these schemes on the basis of number of passes required for encryption and decryption. For a given message M of n blocks, we use the usual notion of a "pass" where 1 pass means we need to store the full length message in the memory and we require an additional pass in case when second pass can not be started until the last data from first pass is completed. In this way RKAE scheme require 1 pass [1] for encryption during execution of Conceal algorithm (as encryption and hashing can be done in parallel i.e. as soon as we get the ciphertext block, pass it to hash function), while the number of passes required for DTM depends on the underlying *AE* scheme. sp-AELM and its variants require only 1 pass for encryption i.e. it visits the n block message exactly once. For decryption and verification, RKAE require 2 passes[1] as shown in Fig 6.2(b) (1 pass for computing the hash value and another one for decryption). DTM uses 2 additional passes for pseudo random number generator (one pass for masking the n block message and another for unmasking the n blocks) apart from the number of passes for the underlying authenticated decryption algorithm. sp-AELM and its variants require only two passes (one at the receiver side and another one at the sender side as shown in Fig. 7.3(b)) for the decryption and verification. We have also used communication overhead between the host and the Crypto module as a comparison parameter in terms of the number of message blocks n. RKAE require only 2 communications(shown in Fig. 6.2(b)), DTM require $2n + 2$ communications(shown in

[1]For RKAE encryption/decryption we neglect the computations carried out at the Crypto module, as it is processed on a small data.

Fig. 6.3(b)) whereas sp-AELM and its variants require $n + 2$ communications(shown in Fig. 7.3(b)).

Parameters	**RKAE** [32]	**DTM** [38]	**sp-AELM** [This paper]	**sp-AELM variants** [This paper]
Online Encryption	Depends on underlying AE	Depends on underlying AE	Yes	Yes
Support for low memory verification	Yes	Yes	Yes	Yes
No. of Passes required for Encryption and tag generation	Two	Depends on underlying AE	One	One
No. of Passes required for Decryption and Verification	Two	2 + Depends on underlying AE	Two	Two
Communications overhead b/w Host and Crypto module during verification and decryption	2	$2n + 2$	$n + 2$	$n + 2$
Knowledge of Plaintext and Ciphertext to Crypto module	No	Yes	Yes	Yes

Table 7.1: Comparison of RKAE, DTM, sp-AELM and sp-AELM variants:(a) One pass is defined as processing of n blocks of a message once.(b)The communication overhead is given in terms of number of blocks n for a message M.

7.7 Summary

In this work, we proposed a new generalized technique for AE schemes to support devices with limited memory. This new technique has been explained in this chapter through a new sponge based AE scheme sp-AELM. sp-AELM can be used to support cryptographic modules having limited storage capabilities. We provided its security proof in an ideal permutation model using the code-based game-playing framework for both privacy and authenticity. In addition to this, we also present two more variants of sp-AELM that serve the same purpose and are more efficient than sp-AELM as they do not require the feed-forward operation as shown in the actual construction of sp-AELM. Hence, we recommend using variant 1 of the sp-AELM construction for applications with resource constraint limitations.

Games for Privacy Proof

Game G0: Initialise $IV_1 = 0^r, IV_2 = 0^c$, $K \xleftarrow{\$} \{0,1\}^r$, $K_r \| K_c \xleftarrow{\$} \{0,1\}^b$, $I_\pi = \{((IV_1 \oplus K) \| IV_2, K_r \| K_c)\}$

On Encryption-Query $(N, A, M, flag)$

1. **if** $(flag = 1)$ **then**
   $M = m_0 \| m_1 \| ..... \| m_{n-1}$, Where
   $|m_i| = r$ and $0 \leq i < (n-1)$
   $Pad(M) =$
   $m_0 \| m_1 \| ..... \| (m_{n-1} \| 10^{r-(|m_{n-1}|+1)})$
   else
   $M = m_0 \| m_1 \| ... \| m_{n-1}$, Where
   $|m_i| = r$
   if $|m_{n-1}| < r$ **then** {return Invalid;}
2. $|Pad(N)| = |Pad(A)| = r$
3. $x = K_r \oplus N$, $w = K_c$
4. $N_r \| N_c = \pi(x \| w)$
5. $x = N_r \oplus A$, $w = N_c \oplus w$
6. $A_r \| A_c = \pi(x \| w)$
7. $x = A_r$,$w = A_c \oplus w$
8. $a_0 = x, b_0 = w$
9. $c_0 = a_0 \oplus m_0$
10. **for** $i = 1 \rightarrow n$ **do**
    $x^{'} \| w^{'} = \pi(c_{i-1} \| b_{i-1})$
    $b_i = b_{i-1} \oplus w^{'}$, $a_i = x^{'}$
    $c_i = a_i \oplus m_i$
11. $C = c_0 \| c_1 \| ..... \| c_n$
12. **if** $(flag = 0)$ **then**
    return C.
13. $x = a_n, w = b_n$
14. $x = x \oplus K$
15. $K_r^{'} \| K_c^{'} = \pi(x \| w)$
16. $x = K_r^{'} \oplus N$, $w = K_c^{'} \oplus w$
17. $N_r \| N_c = \pi(x \| w)$
18. $x = N_r \oplus A$, $w = N_c \oplus w$
19. $A_r \| A_c = \pi(x \| w)$
20. $x = A_r$,$w = A_c \oplus w$
21. $T = z_0 = x$
22. return (C, T)

On π-Query m, where $m \in \{0,1\}^b$

1. let $(x || w) = m$, where $x \in \{0,1\}^r$, $w \in \{0,1\}^c$,
2. if $(m, v) \in I_\pi$ then return v
3. $v \xleftarrow{\$} \{0,1\}^b$
4. if $\exists\ m'$ s.t $(m', v) \in I_\pi$, then
   $v \xleftarrow{\$} \{0,1\}^b \setminus \{v : (*, v) \in I_\pi\}$, where
   $* \in \{0,1\}^b$
5. $I_\pi = I_\pi \bigcup \{(m, v)\}$
6. return v;

On π^{-1}-Query $v = \{v_1 || v_2\}$. where $v_1 \in \{0,1\}^r$, $v_2 \in \{0,1\}^c$, $v \in \{0,1\}^b$

1. if $(m, v) \in I_\pi$ then return m
2. $m \xleftarrow{\$} \{0,1\}^b$
3. if $\exists\ v'$ s.t $(m, v') \in I_\pi$, then
   $m \xleftarrow{\$} \{0,1\}^b \setminus \{m : (m, *) \in I_\pi\}$, where
   $* \in \{0,1\}^b$
4. $I_\pi = I_\pi \bigcup \{(m, v)\}$
5. return m;

Figure 7.6: Game G0

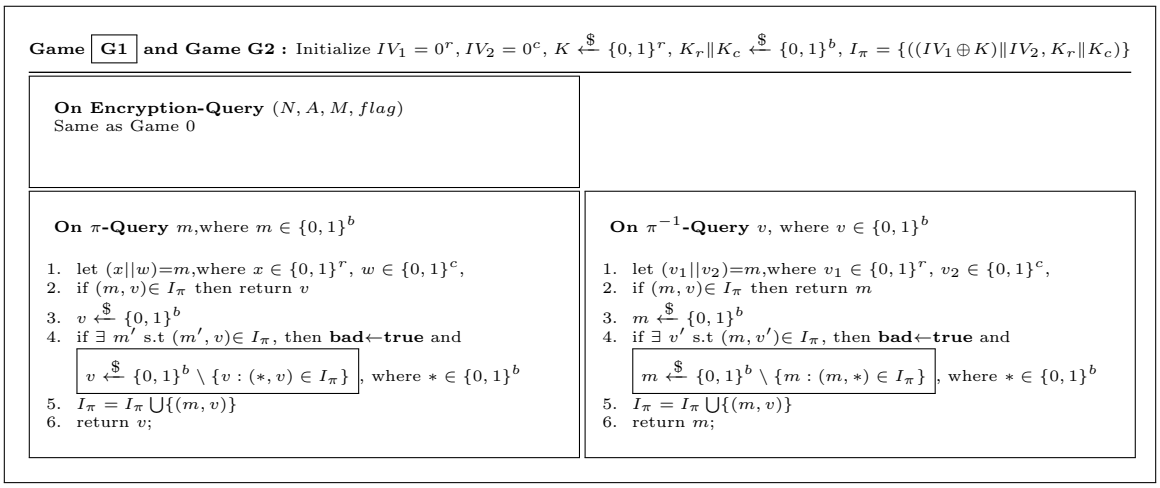


Figure 7.7: Game G1 and Game G2

Game G3 and Game G4 : Initialise $IV_1 = 0^r, IV_2 = 0^c$, $K \xleftarrow{\$} \{0,1\}^r$, $K_r \| K_c \xleftarrow{\$} \{0,1\}^b$, $I_\pi = \{((IV_1 \oplus K) \| IV_2, K_r \| K_c)\}$, $I_c = \{w\}$

On Encryption-Query $(N, A, M, flag)$

1. **if** $(flag = 1)$ **then**
   $M = m_0 \| m_1 \| ..... \| m_{n-1}$, Where $|m_i| = r$ and $0 \leq i < (n-1)$
   $Pad(M) =$
   $m_0 \| m_1 \| ..... \| (m_{n-1} \| 10^{r-(|m_{n-1}|+1)})$
   else
   $M = m_0 \| m_1 \| ... \| m_{n-1}$, Where $|m_i| = r$
   if $|m_{n-1}| < r$ **then** {return Invalid;}
2. $|Pad(N)| = |Pad(A)| = r$
3. $x = IV_1, w = IV_2$
4. $x = K_r \oplus N$,$w = K_c$
5. **if** $\exists v$ *s.t.*$(x \| w, v) \in I_\pi$, **then**
   $N_r \| N_c = v$
   else
   $N_r \| N_c \xleftarrow{\$} \{0,1\}^b$
   if $(N_c \oplus w) \in I_c$ then $bad \leftarrow true$
   $N_c \xleftarrow{\$} \{0,1\}^c \setminus \{N_c^{'} : (N_c^{'} \oplus w) \in I_c\}$
   $I_\pi = I_\pi \cup \{(x \| w, N_r \| N_c)\}$
6. $x = N_r \oplus A$,$w = N_c \oplus w$
7. $I_c = I_c \cup \{w\}$
8. **if** $\exists v$ *s.t.* $(x \| w, v) \in I_\pi$, **then**
   $A_r \| A_c = v$
   else
   $A_r \| A_c \xleftarrow{\$} \{0,1\}^b$
   if $(A_c \oplus w) \in I_c$ then $bad \leftarrow true$
   $A_c \xleftarrow{\$} \{0,1\}^c \setminus \{A_c^{'} : (A_c^{'} \oplus w) \in I_c\}$
   $I_\pi = I_\pi \cup \{(x \| w, A_r \| A_c)\}$
9. $x = A_r$,$w = A_c \oplus w$
10. $I_c = I_c \cup \{w\}$
11. $a_0 = x, b_0 = w$
12. $c_0 = a_0 \oplus m_0$
13. **for** $i = 1 \rightarrow n$ **do**
    if $\exists v$ *s.t.* $(c_{i-1} \| b_{i-1}, v) \in I_\pi$, **then**
    $x^{'} \| w^{'} = v$
    else
    $x^{'} \| w^{'} \xleftarrow{\$} \{0,1\}^b$
    if $(b_{i-1} \oplus w^{'}) \in I_c$ then $bad \leftarrow true$
    $w^{'} \xleftarrow{\$} \{0,1\}^c \setminus \{w^{''} : (b_{i-1} \oplus w^{''}) \in I_c\}$
    $I_\pi = I_\pi \cup \{(c_{i-1} \| b_{i-1}, x^{'} \| w^{'})\}$
    $b_i = b_{i-1} \oplus w^{'}$
    $I_c = I_c \cup \{b_i\}$
    $a_i = x^{'}$
    $c_i = a_i \oplus m_i$
14. $C = c_0 \| c_1 \| ..... \| c_n$
15. **if** $(flag = 0)$ **then**
    return C.
16. $x = a_n, w = b_n$
17. $x = x \oplus K$

Continue...

18. **if** $\exists v$ *s.t.*$(x \| w, v) \in I_\pi$, **then**
    $K_r^{'} \| K_c^{'} = v$
    else
    $K_r^{'} \| K_c^{'} \xleftarrow{\$} \{0,1\}^b$
    if $(w \oplus K_c^{'}) \in I_c$ then $bad \leftarrow true$
    $K_c^{'} \xleftarrow{\$} \{0,1\}^c \setminus \{K_c^{''} : (w \oplus K_c^{''}) \in I_c\}$
    $I_\pi = I_\pi \cup \{(x \| w, K_r^{'} \| K_c^{'})\}$
19. $x = K_r^{'} \oplus N$, $w = K_c^{'} \oplus w$
20. $I_c = I_c \cup \{w\}$
21. Repeat step from 5 to 8.
22. $z_0 = x$
23. $T = z_0$
24. return (C, T)

On π**-Query** m, where $m \in \{0,1\}^b$

1. let $(x \| w) = m$, where $x \in \{0,1\}^r$, $w \in \{0,1\}^c$,
2. **if** $\exists$ $\{v_1, v_2, ...v_t\}$ s.t. $(m, v_i) \in I_\pi$ then return $v \xleftarrow{\$} \{v_1, v_2, ...v_t\}$
3. **else** $v \xleftarrow{\$} \{0,1\}^b$
4. $I_\pi = I_\pi \bigcup \{(m, v)\}$
5. return v;

On π^{-1}**-Query** $v = \{v_1 || v_2\}$. where $v_1 \in \{0,1\}^r$, $v_2 \in \{0,1\}^c$, $v \in \{0,1\}^b$

1. **if** $\exists$ $\{m_1, m_2, ...m_t\}$ s.t. $(m_i, v) \in I_\pi$ then return $m \xleftarrow{\$} \{m_1, m_2, ...vm_t\}$
2. **else** $m \xleftarrow{\$} \{0,1\}^b$
3. $I_\pi = I_\pi \bigcup \{(m, v)\}$
4. return m;

Figure 7.8: Game G3 and Game G4

Game G5 and Game $\boxed{\textbf{G6}}$ **:** Initialise $IV_1 = 0^r, IV_2 = 0^c$, $K \xleftarrow{\$} \{0,1\}^r$, $K_r \| K_c \xleftarrow{\$} \{0,1\}^b$, $I_\pi^{'} = \{((IV_1 \oplus K) \| IV_2, K_r \| K_c)\}$, $I_\pi^{''} = \phi$, $I_\pi = I_\pi^{'} \cup I_\pi^{''}$, $I_c = \{K_c\}$,$I_c^{'} = \phi$

On Encryption-Query $(N, A, M, flag)$

1. **if** $(flag = 1)$ **then**
   $M = m_0 \| m_1 \| ..... \| m_{n-1}$, Where $|m_i| = r$ and $0 \leq i < (n-1)$
   $Pad(M) = m_0 \| m_1 \| ..... \| (m_{n-1} \| 10^{r-(|m_{n-1}|+1)})$
   else
   $M = m_0 \| m_1 \| ... \| m_{n-1}$, Where $|m_i| = r$
   if $|m_{n-1}| < r$ **then** {return Invalid;}
2. $|Pad(N)| = |Pad(A)| = r$
3. $x = IV_1, w = IV_2$
4. $x = K_r \oplus N$,$w = K_c$
5. **if** $\exists v$ *s.t.*$(x \| w, v) \in I_\pi$, **then**
   $\boxed{\textbf{if } \exists v \text{ s.t. } (x \| w, v) \in I_\pi^{''}, \text{ then } bad_1 \leftarrow true}$
   $N_r \| N_c = v$
   else
   $N_r \| N_c \xleftarrow{\$} \{0,1\}^b$
   if $((N_c \oplus w) \in I_c$ ***OR*** $N_c \in I_c^{'})$, **then**
   $N_c \xleftarrow{\$} \{0,1\}^c \setminus \{N_c^{'} : (N_c^{'} \oplus w) \in I_c, N_c^{'} \in I_c^{'}\}$,
   $I_\pi^{'} = I_\pi^{'} \cup \{(x \| w, N_r \| N_c)\}$, $I_\pi = I_\pi \cup I_\pi^{'}$
6. $x = N_r \oplus A$,$w = N_c \oplus w$
7. $I_c = I_c \cup \{w\}$, $I_c^{'} = I_c^{'} \cup \{N_c\}$
8. **if** $\exists v$ *s.t.*$(x \| w, v) \in I_\pi$, **then**
   $\boxed{\textbf{if } \exists v \text{ s.t. } (x \| w, v) \in I_\pi^{''}, \text{ then } bad_2 \leftarrow true}$
   $A_r \| A_c = v$
   else
   $A_r \| A_c \xleftarrow{\$} \{0,1\}^b$
   if $((A_c \oplus w) \in I_c$ ***OR*** $A_c \in I_c^{'})$ **then**
   $A_c \xleftarrow{\$} \{0,1\}^c \setminus \{A_c^{'} : (w \oplus A_c^{'}) \in I_c, A_c^{'} \in I_c^{'}\}$
   $I_\pi^{'} = I_\pi^{'} \cup \{(x \| w, A_r \| A_c)\}$, $I_\pi = I_\pi \cup I_\pi^{'}$
9. $x = A_r$,$w = A_c \oplus w$
10. $I_c = I_c \cup \{w\}$, $I_c^{'} = I_c^{'} \cup \{A_c\}$
11. $a_0 = x, b_0 = w$
12. $c_0 = a_0 \oplus m_0$
13. **for** $i = 1 \rightarrow n$ **do**
    if $\exists v$ *s.t.* $(c_{i-1} \| b_{i-1}, v) \in I_\pi$, **then**
    $\boxed{\textbf{if } \exists v \text{ s.t. } (c_{i-1} \| b_{i-1}, v) \in I_\pi^{''}, \text{ then } bad_2 \leftarrow true}$
    $x^{'} \| w^{'} = v$
    else
    $x^{'} \| w^{'} \xleftarrow{\$} \{0,1\}^b$
    if $((b_{i-1} \oplus w^{'}) \in I_c$ ***OR*** $w' \in I_c^{'})$ **then**
    $w^{'} \xleftarrow{\$} \{0,1\}^c \setminus \{w'' : (b_{i-1} \oplus w'') \in I_c, w'' \in I_c^{'}\}$
    $I_\pi^{'} = I_\pi^{'} \cup \{(c_{i-1} \| b_{i-1}, x^{'} \| w^{'})\}$, $I_\pi = I_\pi \cup I_\pi^{'}$
    $b_i = b_{i-1} \oplus w^{'}$
    $I_c = I_c \cup \{b_i\}$, $I_c^{'} = I_c^{'} \cup \{w'\}$
    $a_i = x^{'}$, $c_i = a_i \oplus m_i$
14. $C = c_0 \| c_1 \| ..... \| c_n$
15. **if** $(flag = 0)$ **then**
    return C.
16. $x = a_n, w = b_n$
17. $x = x \oplus K$

Continue..

18. **if** $\exists v$ *s.t.* $(x \| w, v) \in I_\pi$, **then**
    $\boxed{\textbf{if } \exists v \text{ s.t. } (x \| w, v) \in I_\pi^{''}, \text{ then } bad_2 \leftarrow true}$
    $K_r^{'} \| K_c^{'} = v$
    else
    $K_r^{'} \| K_c^{'} \xleftarrow{\$} \{0,1\}^b$
    if $((w \oplus K_c^{'}) \in I_c$ ***OR*** $w' \in I_c^{'})$, **then**
    $K_c^{'} \xleftarrow{\$} \{0,1\}^c \setminus \{K_c^{''} : (w \oplus K_c^{''}) \in I_c,\ K_c^{''} \in I_c^{'}\}$
    $I_\pi^{'} = I_\pi^{'} \cup \{(x \| w, K_r^{'} \| K_c^{'})\}$, $I_\pi = I_\pi \cup I_\pi^{'}$
19. $x = K_r^{'} \oplus N$, $w = K_c^{'} \oplus w$
20. $I_c = I_c \cup \{w\}$, $I_c^{'} = I_c^{'} \cup \{K_c^{'}\}$
21. Repeat step from 5 to 8.
22. $z_0 = x$
23. $T = z_0$
24. return (C, T)

On π**-Query** m, where $m \in \{0,1\}^b$

1. let $(x || w) = m$, where $x \in \{0,1\}^r$, $w \in \{0,1\}^c$,
2. **if** $\exists v$ s.t. $(x \| w, v) \in I_\pi^{'}$, then $\boxed{bad \leftarrow true}$
   return v;
3. **if** $\exists \{v_1, v_2, ...v_t\}$ s.t. $(m, v_i) \in I_\pi^{''}$ then
   return $v \xleftarrow{\$} \{v_1, v_2, ...v_t\}$
4. **else** $v \xleftarrow{\$} \{0,1\}^b$
5. $I_\pi^{''} = I_\pi^{''} \bigcup \{(m, v)\}$
6. $I_\pi = I_\pi \cup I_\pi^{''}$
7. return v;

On π^{-1}**-Query** $v = \{v_1 || v_2\}$. where $v_1 \in \{0,1\}^r$, $v_2 \in \{0,1\}^c$, $v \in \{0,1\}^b$

1. **if** $\exists m$ s.t. $(m, v) \in I_\pi^{'}$, then $\boxed{bad \leftarrow true}$
   return m
2. **if** $\exists \{m_1, m_2, ...m_t\}$ s.t. $(m_i, v) \in I_\pi^{''}$ then
   return $m \xleftarrow{\$} \{m_1, m_2, ..., m_t\}$
3. **else** $m \xleftarrow{\$} \{0,1\}^b$
4. $I_\pi^{''} = I_\pi^{''} \bigcup \{(m, v)\}$
5. $I_\pi = I_\pi \cup I_\pi^{''}$
6. return m;

Figure 7.9: Game G5 and Game G6

Game G7: Initialise $I_\pi = \emptyset$, $IV_1 = 0^r$ $IV_2 = 0^c$, $K \xleftarrow{\$} \{0,1\}^r$, $I_m = \phi$

On Encryption-Query $(N, A, M, flag)$

1. **if** $(flag = 1)$ **then**
   - $M = m_0 \| m_1 \| ..... \| m_{n-1}$, Where $|m_i| = r$ and $0 \leq i < (n-1)$
   - $Pad(M) =$ $m_0 \| m_1 \| ..... \| (m_{n-1} \| 10^{r-(|m_{n-1}|+1)})$

   else
   - $M = m_0 \| m_1 \| ... \| m_{n-1}$, Where $|m_i| = r$
   - **if** $|m_{n-1}| < r$ **then** {return Invalid;}
2. $|Pad(N)| = |Pad(A)| = r$
3. $x = IV_1, w = IV_2$
4. **if** $(flag = 0)$ **then**
   - **if** $\exists C'$ $s.t. (N, A, m_0 \| \dots \| m_j, C') \in I_m$ *where* $0 \leq j < (n-1)$ **then**
     - $c_0 \| \dots \| c_j \leftarrow C'$
     - **for** $i = (j+1) \rightarrow (n-1)$ **do**
       - $c_i \xleftarrow{\$} \{0,1\}^r$
     - $C = c_0 \| \dots \| c_j \| c_{j+1} \dots \| c_{n-1}$

   else
   - **for** $i = 0 \rightarrow n-1$ **do**
     - $c_i \xleftarrow{\$} \{0,1\}^r$
   - $C = c_0 \| \dots \| c_{n-1}$

   $I_m = I_m \cup \{(N, A, M, C)\}$
   return C
5. **for** $i = 0 \rightarrow n-1$ **do**
   - $c_i \xleftarrow{\$} \{0,1\}^r$
6. $C = c_0 \| \dots \| c_{n-1}$
7. $T \xleftarrow{\$} \{0,1\}^r$
8. return (C, T)

On π-Query m, where $m \in \{0,1\}^b$

1. let $(x \| w) = m$, where $x \in \{0,1\}^r$, $w \in \{0,1\}^c$,
2. if $(m, v) \in I_\pi$ then return v
3. $v \xleftarrow{\$} \{0,1\}^b$
4. if $\exists$ m' s.t $(m', v) \in I_\pi$, then $v \xleftarrow{\$} \{0,1\}^b \setminus \{v : (*, v) \in I_\pi\}$, where $* \in \{0,1\}^b$
5. $I_\pi = I_\pi \bigcup \{(m, v)\}$
6. return v;

On π^{-1}-Query $v = \{v_1 \| v_2\}$. where $v_1 \in \{0,1\}^r$, $v_2 \in \{0,1\}^c$, $v \in \{0,1\}^b$

1. if $(m, v) \in I_\pi$ then return m
2. $m \xleftarrow{\$} \{0,1\}^b$
3. if $\exists$ v' s.t $(m, v') \in I_\pi$, then $m \xleftarrow{\$} \{0,1\}^b \setminus \{m : (m, *) \in I_\pi\}$, where $* \in \{0,1\}^b$
4. $I_\pi = I_\pi \bigcup \{(m, v)\}$
5. return m;

Figure 7.10: Game G7

Games for Authenticity Proof

Game $G0'$**:** Initialize $IV_1 = 0^r, IV_2 = 0^c$, $K \xleftarrow{\$} \{0,1\}^r$, $K_r \| K_c \xleftarrow{\$} \{0,1\}^b$

On Encryption-Query $(N, A, M, flag)$

1. **if** $(flag = 1)$ **then**
   $M = m_0 \| m_1 \| ..... \| m_{n-1}$, Where $|m_i| = r$ and $0 \leq i < (n-1)$
   $Pad(M) =$ $m_0 \| m_1 \| ..... \| (m_{n-1} \| 10^{r-(|m_{n-1}|+1)})$
   else
   $M = m_0 \| m_1 \| ... \| m_{n-1}$, Where $|m_i| = r$
   if $|m_{n-1}| < r$ **then** {return Invalid;}
2. $|Pad(N)| = |Pad(A)| = r$
3. $x = K_r \oplus N$, $w = K_c$
4. $N_r \| N_c = \pi(x \| w)$
5. $x = N_r \oplus A$, $w = N_c \oplus w$
6. $A_r \| A_c = \pi(x \| w)$
7. $x = A_r$,$w = A_c \oplus w$
8. $a_0 = x, b_0 = w$
9. $c_0 = a_0 \oplus m_0$
10. **for** $i = 1 \rightarrow n$ **do**
    $x^{'} \| w^{'} = \pi(c_{i-1} \| b_{i-1})$
    $b_i = b_{i-1} \oplus w^{'}$
    $a_i = x^{'}$
    $c_i = a_i \oplus m_i$
11. $C = c_0 \| c_1 \| ..... \| c_n$
12. $x = a_n, w = b_n$
13. $x = x \oplus K$
14. $K_r^{'} \| K_c^{'} = \pi(x \| w)$
15. $x = K_r^{'} \oplus N$, $w = K_c^{'} \oplus w$
16. $N_r \| N_c = \pi(x \| w)$
17. $x = N_r \oplus A$, $w = N_c \oplus w$
18. $A_r \| A_c = \pi(x \| w)$
19. $x = A_r$,$w = A_c \oplus w$
20. $z_0 = x$
21. $T = z_0$
22. Return (C, T)

On Decryption-Query (N, A, C, T)

1. $Pad(C) = c_0 \| c_1 \| ..... \| c_n$, Where $|c_i| = r$
2. $|Pad(N)| = |Pad(A)| = r$
3. $x = K_r \oplus N$, $w = K_c$
4. $N_r \| N_c = \pi(x \| w)$
5. $x = N_r \oplus A$, $w = N_c \oplus w$
6. $A_r \| A_c = \pi(x \| w)$
7. $x = A_r$,$w = A_c \oplus w$
8. $a_0 = x, b_0 = w$
9. $m_0 = a_0 \oplus c_0$
10. **for** $i = 1 \rightarrow n$ **do**
    $x^{'} \| w^{'} = \pi(c_{i-1} \| b_{i-1})$
    $b_i = b_{i-1} \oplus w^{'}$
    $a_i = x^{'}$
    $m_i = a_i \oplus c_i$
11. $M = m_0 \| m_1 \| ..... \| m_n$
12. $x = a_n, w = b_n$
13. $x = x \oplus K$
14. $K_r^{'} \| K_c^{'} = \pi(x \| w)$
15. $x = K_r^{'} \oplus N$, $w = K_c^{'} \oplus w$
16. $N_r \| N_c = \pi(x \| w)$
17. $x = N_r \oplus A$, $w = N_c \oplus w$
18. $A_r \| A_c = \pi(x \| w)$
19. $x = A_r$,$w = A_c \oplus w$
20. $z_0 = x$
21. **if** $(z_0 == T)$ **then**
    Return (a_0, b_0)
    else
    Return $\perp$

On π**-Query** m, where $m \in \{0,1\}^b$

1. let $(x||w) = m$, where $x \in \{0,1\}^r$, $w \in \{0,1\}^c$,
2. if $(m, v) \in I_\pi$ then return v
3. $v \xleftarrow{\$} \{0,1\}^b$
4. if $\exists\ m'$ s.t $(m', v) \in I_\pi$, then $v \xleftarrow{\$} \{0,1\}^b \setminus \{v : (*, v) \in I_\pi\}$, where $* \in \{0,1\}^b$
5. $I_\pi = I_\pi \bigcup \{(m, v)\}$
6. return v;

On π^{-1}**-Query** $v = \{v_1||v_2\}$. where $v_1 \in \{0,1\}^r,\ v_2 \in \{0,1\}^c,\ v \in \{0,1\}^b$

1. if $(m, v) \in I_\pi$ then return m
2. $m \xleftarrow{\$} \{0,1\}^b$
3. if $\exists\ v'$ s.t $(m, v') \in I_\pi$, then $m \xleftarrow{\$} \{0,1\}^b \setminus \{m : (m, *) \in I_\pi\}$, where $* \in \{0,1\}^b$
4. $I_\pi = I_\pi \bigcup \{(m, v)\}$
5. return m;

Figure 7.11: Game $G0'$

Game $\boxed{G1'}$ **and Game** $G2'$ **:** Initialise $IV_1 = 0^r, IV_2 = 0^c$, $K \xleftarrow{\$} \{0,1\}^r$, $K_r \| K_c \xleftarrow{\$} \{0,1\}^b$, $I_\pi = \{((IV_1 \oplus K) \| IV_2, K_r \| K_c)\}$

On Encryption-Query $(N, A, M, flag)$
Same as Game 0

On Decryption-Query (N, A, C, T)
Same as Game 0

On π-Query m,where $m \in \{0,1\}^b$

1. let $(x||w)$=m,where $x \in \{0,1\}^r$, $w \in \{0,1\}^c$,
2. if $(m, v) \in I_\pi$ then return v
3. $v \xleftarrow{\$} \{0,1\}^b$
4. if $\exists\ m'$ s.t $(m', v) \in I_\pi$, then **bad**$\leftarrow$**true** and $\boxed{v \xleftarrow{\$} \{0,1\}^b \setminus \{v : (*, v) \in I_\pi\}}$, where $* \in \{0,1\}^b$
5. $I_\pi = I_\pi \bigcup \{(m, v)\}$
6. return v;

On π^{-1}-Query v, where $v \in \{0,1\}^b$

1. let $(v_1||v_2)$=m,where $v_1 \in \{0,1\}^r$, $v_2 \in \{0,1\}^c$,
2. if $(m, v) \in I_\pi$ then return m
3. $m \xleftarrow{\$} \{0,1\}^b$
4. if $\exists\ v'$ s.t $(m, v') \in I_\pi$, then **bad**$\leftarrow$**true** and $\boxed{m \xleftarrow{\$} \{0,1\}^b \setminus \{m : (m, *) \in I_\pi\}}$, where $* \in \{0,1\}^b$
5. $I_\pi = I_\pi \bigcup \{(m, v)\}$
6. return m;

Figure 7.12: Game $G1'$ and Game $G2'$

Game $G3'$ and Game $\boxed{G4'}$: Initialise $IV_1 = 0^r, IV_2 = 0^c$, $K \xleftarrow{\$} \{0,1\}^r$, $K_r \| K_c \xleftarrow{\$} \{0,1\}^b$, $I_\pi = \{((IV_1 \oplus K) \| IV_2, K_r \| K_c)\}$, $I_c = \{w\}$

On Encryption-Query $(N, A, M, flag)$

1. **if** $(flag = 1)$ **then**
   $M = m_0 \| m_1 \| ..... \| m_{n-1}$, Where $|m_i| = r$ and $0 \leq i < (n-1)$
   $Pad(M) = m_0 \| m_1 \| ..... \| (m_{n-1} \| 10^{r-(|m_{n-1}|+1)})$
   else
   $M = m_0 \| m_1 \| ... \| m_{n-1}$, Where $|m_i| = r$
   if $|m_{n-1}| < r$ **then** {return Invalid;}
2. $|Pad(N)| = |Pad(A)| = r$
3. $x = IV_1, w = IV_2$
4. $x = K_r \oplus N$,$w = K_c$
5. **if** $\exists v$ *s.t.*$(x \| w, v) \in I_\pi$, **then**
   $N_r \| N_c = v$
   else
   $N_r \| N_c \xleftarrow{\$} \{0,1\}^b$
   $\boxed{\text{if } (N_c \oplus w) \in I_c \text{ then } bad \leftarrow true}$
   $\boxed{N_c \xleftarrow{\$} \{0,1\}^c \setminus \{N_c^{'} : (N_c^{'} \oplus w) \in I_c\}}$
   $I_\pi = I_\pi \cup \{(x \| w, N_r \| N_c)\}$
6. $x = N_r \oplus A$,$w = N_c \oplus w$
7. $I_c = I_c \cup \{w\}$
8. **if** $\exists v$ *s.t.* $(x \| w, v) \in I_\pi$, **then**
   $A_r \| A_c = v$
   else
   $A_r \| A_c \xleftarrow{\$} \{0,1\}^b$
   $\boxed{\text{if } (A_c \oplus w) \in I_c \text{ then } bad \leftarrow true}$
   $\boxed{A_c \xleftarrow{\$} \{0,1\}^c \setminus \{A_c^{'} : (A_c^{'} \oplus w) \in I_c\}}$
   $I_\pi = I_\pi \cup \{(x \| w, A_r \| A_c)\}$
9. $x = A_r$,$w = A_c \oplus w$
10. $I_c = I_c \cup \{w\}$
11. $a_0 = x, b_0 = w$
12. $c_0 = a_0 \oplus m_0$
13. **for** $i = 1 \rightarrow n$ **do**
    if $\exists v$ *s.t.* $(c_{i-1} \| b_{i-1}, v) \in I_\pi$, **then**
    $x^{'} \| w^{'} = v$
    else
    $x^{'} \| w^{'} \xleftarrow{\$} \{0,1\}^b$
    $\boxed{\text{if } (b_{i-1} \oplus w^{'}) \in I_c \text{ then } bad \leftarrow true}$
    $\boxed{w^{'} \xleftarrow{\$} \{0,1\}^c \setminus \{w^{''} : (b_{i-1} \oplus w^{''}) \in I_c\}}$
    $I_\pi = I_\pi \cup \{(c_{i-1} \| b_{i-1}, x^{'} \| w^{'})\}$
    $b_i = b_{i-1} \oplus w^{'}$
    $I_c = I_c \cup \{b_i\}$
    $a_i = x^{'}$
    $c_i = a_i \oplus m_i$
14. $C = c_0 \| c_1 \| ..... \| c_n$
15. $x = a_n, w = b_n$
16. $x = x \oplus K$
17. **if** $\exists v$ *s.t.*$(x \| w, v) \in I_\pi$, **then**
    $K_r^{'} \| K_c^{'} = v$
    else
    $K_r^{'} \| K_c^{'} \xleftarrow{\$} \{0,1\}^b$
    $\boxed{\text{if } (w \oplus K_c^{'}) \in I_c \text{ then } bad \leftarrow true}$
    $\boxed{K_c^{'} \xleftarrow{\$} \{0,1\}^c \setminus \{K_c^{''} : (w \oplus K_c^{''}) \in I_c\}}$
    $I_\pi = I_\pi \cup \{(x \| w, K_r^{'} \| K_c^{'})\}$
18. $x = K_r^{'} \oplus N$, $w = K_c^{'} \oplus w$
19. $I_c = I_c \cup \{w\}$
20. Repeat step from 5 to 9.
21. $z_0 = x$
22. $T = z_0$
23. Return (C, T)

On Decryption-Query (N, A, C, T)

1. $Pad(C) = c_0 \| c_1 \| ..... \| c_n$, Where $|c_i| = r$
2. $|Pad(N)| = |Pad(A)| = r$
3. $x = K_r \oplus N$, $w = K_c$
4. **if** $\exists v$ *s.t.*$(x \| w, v) \in I_\pi$, **then**
   $N_r \| N_c = v$
   else
   $N_r \| N_c \xleftarrow{\$} \{0,1\}^b$
   $\boxed{\text{if } (N_c \oplus w) \in I_c \text{ then } bad \leftarrow true}$
   $\boxed{N_c \xleftarrow{\$} \{0,1\}^c \setminus \{N_c^{'} : (N_c^{'} \oplus w) \in I_c\}}$
   $I_\pi = I_\pi \cup \{(x \| w, N_r \| N_c)\}$
5. $x = N_r \oplus A$, $w = N_c \oplus w$
6. $I_c = I_c \cup \{w\}$
7. **if** $\exists v$ *s.t.* $(x \| w, v) \in I_\pi$, **then**
   $A_r \| A_c = v$
   else
   $A_r \| A_c \xleftarrow{\$} \{0,1\}^b$
   $\boxed{\text{if } (A_c \oplus w) \in I_c \text{ then } bad \leftarrow true}$
   $\boxed{A_c \xleftarrow{\$} \{0,1\}^c \setminus \{A_c^{'} : (A_c^{'} \oplus w) \in I_c\}}$
   $I_\pi = I_\pi \cup \{(x \| w, A_r \| A_c)\}$
8. $x = A_r$,$w = A_c \oplus w$
9. $I_c = I_c \cup \{w\}$
10. $a_0 = x, b_0 = w$
11. $m_0 = a_0 \oplus c_0$
12. **for** $i = 1 \rightarrow n$ **do**
    if $\exists v$ *s.t.* $(c_{i-1} \| b_{i-1}, v) \in I_\pi$, **then**
    $x^{'} \| w^{'} = v$
    else
    $x^{'} \| w^{'} \xleftarrow{\$} \{0,1\}^b$
    $\boxed{\text{if } (b_{i-1} \oplus w^{'}) \in I_c \text{ then } bad \leftarrow true}$
    $\boxed{w^{'} \xleftarrow{\$} \{0,1\}^c \setminus \{w^{''} : (b_{i-1} \oplus w^{''}) \in I_c\}}$
    $I_\pi = I_\pi \cup \{(c_{i-1} \| b_{i-1}, x^{'} \| w^{'})\}$
    $b_i = b_{i-1} \oplus w^{'}$
    $I_c = I_c \cup \{b_i\}$
    $a_i = x^{'}$
    $m_i = a_i \oplus c_i$
13. $M = m_0 \| m_1 \| ..... \| m_n$
14. $x = a_n, w = b_n$
15. $x = x \oplus K$
16. **if** $\exists v$ *s.t.*$(x \| w, v) \in I_\pi$, **then**
    $K_r^{'} \| K_c^{'} = v$
    else
    $K_r^{'} \| K_c^{'} \xleftarrow{\$} \{0,1\}^b$
    $\boxed{\text{if } (w \oplus K_c^{'}) \in I_c \text{ then } bad \leftarrow true}$
    $\boxed{K_c^{'} \xleftarrow{\$} \{0,1\}^c \setminus \{K_c^{''} : (w \oplus K_c^{''}) \in I_c\}}$
    $I_\pi = I_\pi \cup \{(x \| w, K_r^{'} \| K_c^{'})\}$
17. $x = K_r^{'} \oplus N$, $w = K_c^{'} \oplus w$
18. $I_c = I_c \cup \{w\}$
19. Repeat step from 4 to 8.
20. $z_0 = x$
21. **if** $(z_0 == T)$ **then**
    Return (a_0, b_0)
    else
    Return $\perp$

Figure 7.13: Game G3' and Game G4'

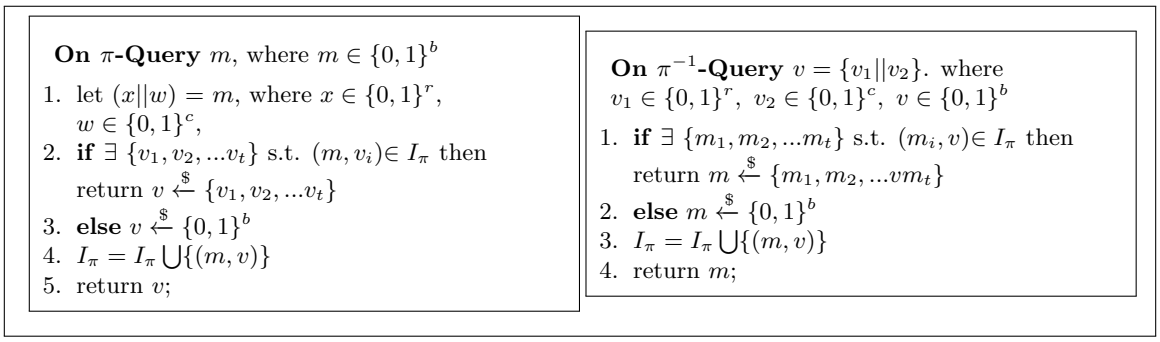


Figure 7.14: Game $G3'$ and Game $G4'$

Chapter 8

Analysis of Sponge based Submissions in CAESAR using Limited Memory Technique

In this chapter, we apply our newly proposed generalized technique, which stores only one intermediate state instead of entire plaintext during tag verification, on various sponge based schemes submitted to the CAESAR competition to determine their suitability for supporting devices having limited memory constraint. There are ten sponge based AE schemes submitted to CAESAR for the first round, out of which one scheme (CBEAM [56]) has been withdrawn. Currently there are nine sponge based *AE* schemes competing for the next round. In the following subsections we present the brief analysis of each of these 9 schemes after applying the same technique used in sp-AELM.

8.1 ARTEMIA [49]

Artemia is family of dedicated authenticated encryption scheme. It has two variants Artemia-256 which uses 512 bit permutation and Artemia-128 which uses 256 bit permutation in the JHAE mode(shown in Fig. 8.1).

While analyzing this mode, we find that it can not support low memory device constraint. Our analysis is based on the same technique proposed in sp-AELM i.e., storing only one intermediate state(shown using red line in Fig. 8.1) instead of storing whole message during decryption, can not be applied here. Using this intermediate state attacker can find the value of key as he already knows the value of T and can do the forward computation using C_i^s to get the state value after last π block, XORing this value with T will result in value of K.

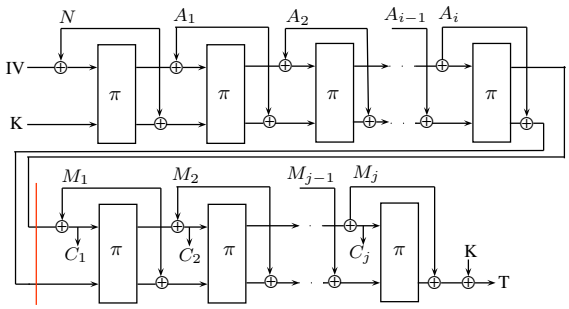


Figure 8.1: JHAE Mode

8.2 ASCON [25]

ASCON is a family of authenticated encryption designs $ASCON_{a,b}-k$. The family members are parameterized by the key length $k \leq 128$ and internal round numbers a and b. Each design specifies an authenticated encryption algorithm $\mathcal{E}_{k,a,b}$ and decryption algorithm $\mathcal{D}_{k,a,b}$.

On analyzing this scheme, we find out that it can support low memory devices by storing only one intermediate value(shown using red line in Fig. 8.2) instead of storing all decrypted blocks during decryption, without breaking the security of construction. This is due to the use of key at the end, which also prevents attcacker from doing forgery. Further, this intermediate value can be used to decrypt the message at user side.

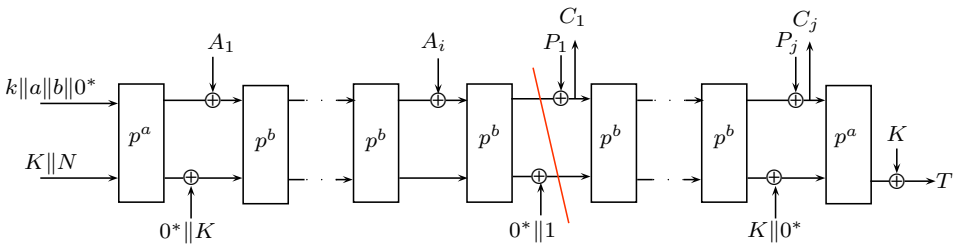


Figure 8.2: ASCON

8.3 ICEPOLE [62]

ICEPOLE is a family of authenticated ciphers with three parameters: key length, secret message number legth, nonce length. The construction of ICEPOLE is shown in Fig. 8.3.

Here, if we store intermediate state(shown using red line in Fig. 8.3) instead of whole decrypted text blocks to support low memory feature, it will result in revealing the key value, as attacker can do calculation in reverse direction to get the actual value of key.

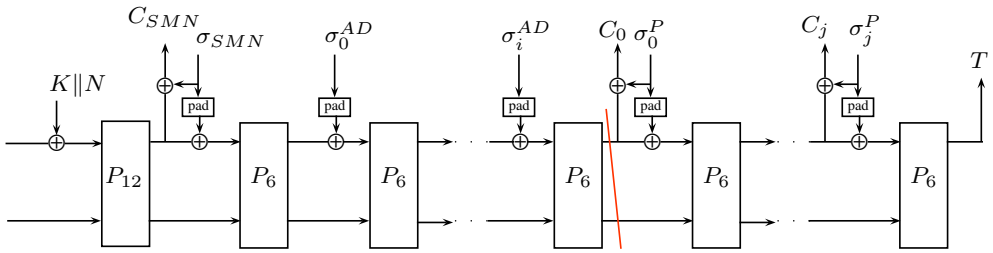


Figure 8.3: ICEPOLE

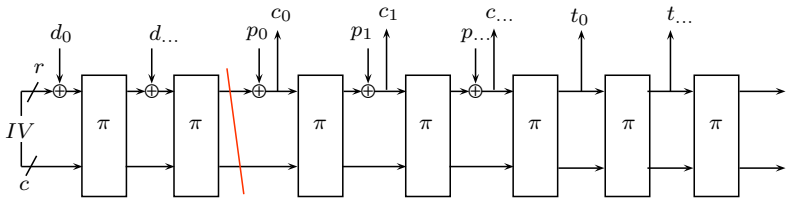


Figure 8.4: STRIBOB. Key, nonce and padded associated data are represented by d_i and tag is givenn by t_i.

8.4 STRIBOB [57]

STRIBOB is an algorithm for authenticated encryption with associated data. It is built on basic sponge mode of authenticated encryption(shown in Fig. 8.4).It uses a 512 512 bit permutation π as its cryptographic foundation. Π in turn is built from 12 iterations of LPS transformation, interleaved with exclusive-or operation with round constants.

In STRIBOB, if we store only intermediate state(shown using red line in Fig. 8.4), instead of storing all $P_i^{'}s$, attacker can get the actual key value by using inverse permutation.

8.5 Π-Cipher [29]

Π-Cipher is parallel, incremental, nonce based, tag second-preimage resistant, authenticated encryption cipher with associated data. Its construction is shown in Fig. 8.5.

This AE scheme can not support limited memory constraint. Suppose if we store CIS'' and $ctr + i + 2$ values instead of storing all decrypted text blocks. Now using these intermediate values and known ciphertext values attacker can calculate the plaintext and Tag T''. However using this T'' he can not get the key due to the unavailability of Secret message number(SMN). Though it is secure against key recovery attack but forgery is possible here. As using T'', it is easy to generate a new ciphertext C and its corresponding valid tag T using a repeated nonce(SMN). This is due to the reason that T'' value will be same for a given key K and secret message number SMN.

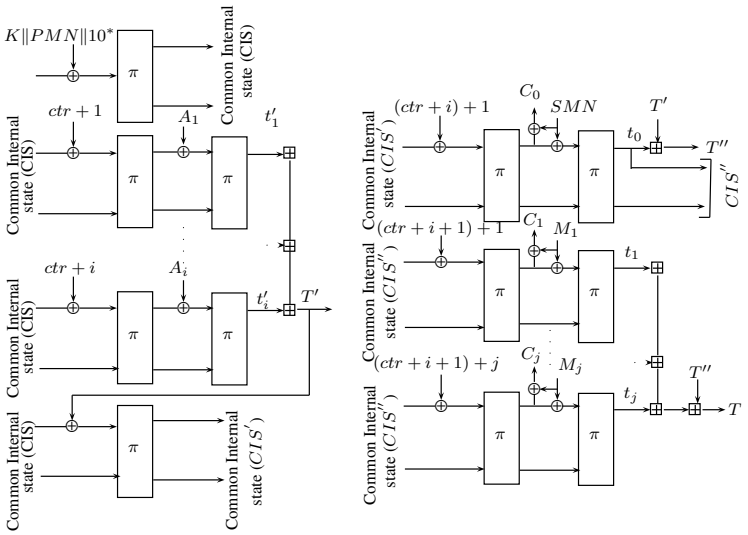


Figure 8.5: II Cipher

8.6 PRIMATE [36]

The authenticated encryption family PRIMATEs is defined by two parameters: the security level $s \in \{10, 15\}$ and mode of operation scheme $\in$ {GIBBON, HANUMAN, APE}. These modes are shown in Fig. 8.6, 8.7, 8.8.

Out of these three modes of operation of PRIMATE authenticated encryption family, only GIBBON can support low memory feature by storing only one intermediate state(shown using red line in Fig. 8.6), without revealing key to the attacker. HANUMAN can be easily broken to get the key, when storing only intermediate state as T is known to the attacker and can do the forward computation to get the state after last permutation block, XORing this value with T will result in finding key. Also, attacker will not be able to create forgery due to the use of key at the end. Similarly APE can also be broken, if we implement the low memory feature in it. As the intermediate state can be used to calculate the key by just applying permutation once and then XORing the last $|T|$ bits with known tag value T.

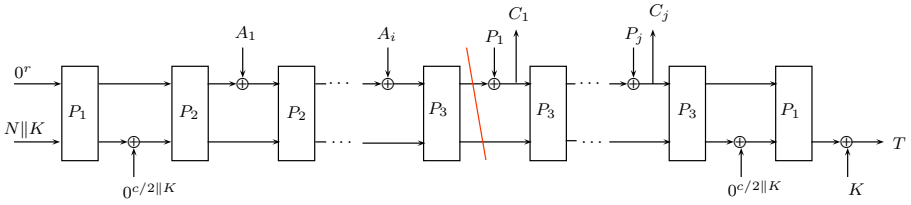


Figure 8.6: GIBBON

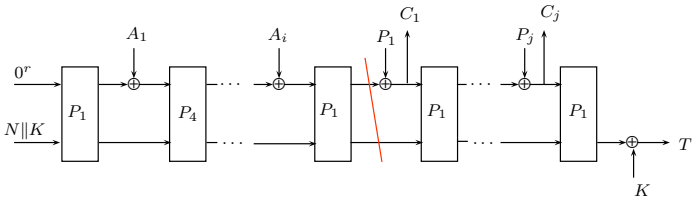


Figure 8.7: HANUMAN

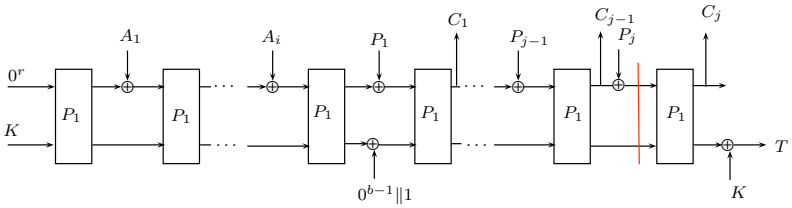


Figure 8.8: APE

8.7 NORX [50]

NORX is an authenticated encryption scheme supporting an arbitrary parallelism degree, based on ARX primitives yet not using modular additions. NORX has a unique parallel architecture based on the monkeyDuplex construction [40]. Fig. 8.9 represents layout of NORX corresponding to $D = 1$. Here also, revealing the intermediate state(shown using red line in Fig. 8.9) will result in finding value of key by doing computation in reverse direction using inverse permutation, breaking the security of construction.

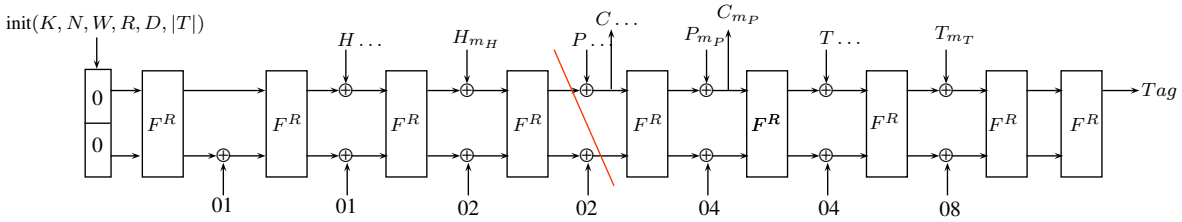


Figure 8.9: NORX for D=1

8.8 Ketje [41]

Ketje is a set of two authenticated encryption functions with support for message associated data. Ketje builds on round-reduced versions of Keccak-f[400] and Keccak-f[200].The construction calling these permutations is MonkeyDuplex [40], a variant of the duplex

construction [16]. The mode that runs on top of MonkeyDuplex is called MonkeyWrap. The MonkeyWrap differ from SpongeWrap as it is built on MonkeyDuplex construction rather than on Duplex, it updates a whole state of b bits using key and nonce together instead of updating only bitrate part (r bits) in SpongwWrap and uses a different MonkeyDuplex call when transitioning to tag generation phase.

Implementation of low memory constraint is not possible for this scheme. As shown in the Fig. 8.10, storing intermediate state(shown using red line in Fig. 8.10), will result in finding value of key K using inverse permutation.

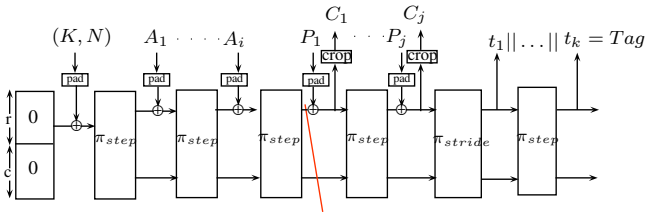


Figure 8.10: MonkeyWrap mode of Authenticated Encryption

8.9 Keyak [42]

Keyak is a set of four authenticated encryption functions with support for message associated data. It builds on round-reduced versions of the Keyak-f [800] and Keyak-f [1600] permutations. It uses the duplex construction [14] on top of one of these permutations. The mode that runs on duplex construction is the DuplexWrap which is almost similar to SpongeWrap. In addition, DuplexWrap defines an explicit forget call(calling it is optional) to ensure forward secrecy. Because of this forget call it is not possible to get the key from the intermediate state. But the forgery attack is possible by generating valid ciphertext tag pair for a different message using same key and nonce.

8.10 Summary

In this chapter, we applied newly proposed intermediate state based technique in previous chapter to all Sponge based AE schemes submitted to the CAESAR competition for determining their suitability to support devices with memory constraint. Our analysis shows that ASCON and one of the PRIMATEs instance namely GIBBON, can support limited memory constraint using this technique, while remaining schemes are not directly suitable for this scenario.

Chapter 9

dAELM: Deterministic Authenticated Encryption for Limited Memory Devices

The concept of the DAE scheme was introduced by Rogaway and Shrimpton in [68]. In a DAE scheme, the encryption algorithm deterministically produces a ciphertext for a given key, associated data, and a message. The DAE construction in [68] was termed "SIV," which stands for synthetic IV. This construction combines a conventional IV-based encryption scheme and a pseudorandom function that takes a vector of strings as input. The scheme [68] first applies a PRF to associated data and a message to produce an IV and uses the resultant IV for the encryption scheme. The SIV construction is proven to be secure, assuming all of its components are secure. However, the SIV construction is not designed for resource-constrained environments; in particular, it may not be suitable for a scenario where the cryptomodule does not have enough space to store the complete message M during decryption. This memory requirement is because the decryption procedure cannot return the plaintext before successful verification of the tag. In this work, we design an efficient and provably secure DAE scheme that is suitable for memory-constrained cryptomodules.

Our scheme resembles the SIV construction and exploits the existing entropy in messages to omit the overhead for nonces. It also saves bandwidth by eliminating the need to send nonces or random values over the communication channel.

The important feature that makes our construction differ from SIV is the use of the cryptographic module. In dAELM, we assume encryption and decryption takes place inside a cryptographic module which has a secret key embedded on it and has a very limited memory space. The reason behind using the cryptographic module is to increase the security of the key. We do not want the key to being accessible outside the cryptographic module.

9.1 Preliminaries

Definition 11 (DAE). *A deterministic AE scheme [68] is a tuple* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$*, where* $\mathcal{K}$ *is a non-empty set of strings and the encryption algorithm* $\mathcal{E}$ *and decryption algorithm* $\mathcal{D}$ *are deterministic.* $\mathcal{E} : \mathcal{K} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C}$ *and* $\mathcal{D} : \mathcal{K} \times \mathcal{A} \times \mathcal{C} \to \mathcal{M} \cup \perp$ *with associated non-empty key space* $\mathcal{K}$*, associated-data space* $\mathcal{A}$*, and message/ciphertext space* $\mathcal{M}, \mathcal{C} \subseteq \{0,1\}^*$*. For each* $K \in \mathcal{K}$*,* $A \in \mathcal{A}$ *and* $M \in \mathcal{M}$*,* $\mathcal{E}_K^A(M)$ *or* $\mathcal{E}_K(A, M)$ *maps* (A, M) *to an output* C *such that* $|C| = |M| + \tau$ *for a fixed* τ*.* $\mathcal{D}_K^A(C)$ *or* $\mathcal{D}_K(A, C)$ *outputs corresponding message* M *iff* C *is valid, and* $\perp$ *otherwise. We assume correctness; i.e., for all* $(K, A, M) \in \mathcal{K} \times \mathcal{A} \times \mathcal{M}$*, it holds that* $\mathcal{D}_K^A(\mathcal{E}_K^A(M)) = M$*.*

Definition 12 (Deterministic Privacy (detPriv)). *We adapt the same notion of privacy as in [68], namely in terms of indistinguishability of the output from a random string. Let* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *be a DAE scheme with a fixed associated data space* $\mathcal{A}$ *and message space* $\mathcal{M}$*. Let* A *be an computationally bounded adversary having access to encryption oracle* $\mathcal{E}$ *and an ideal primitive* π *and its inverse* π^{-1}*. We then define detPriv advantage of adversary* A *in attacking* Π *as*

$$Adv_{\Pi}^{detPriv}(A) = \left| Pr[K \xleftarrow{\$} \mathcal{K} : A^{\mathcal{E}_K(\cdot,\cdot),\pi,\pi^{-1}} = 1] - Pr[A^{\$(\cdot,\cdot),\pi,\pi^{-1}} = 1] \right|$$

where $\$$ *is an oracle which on input* $(A, M) \in \mathcal{A} \times \mathcal{M}$ *outputs a random string of length* $|\mathcal{E}_K(\cdot,\cdot)|$*. We assume that adversary* A *does not allow repeated queries on the same inputs.*

Definition 13 (Deterministic Authenticity (detAuth)). *Let* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *be a DAE scheme with message space* $\mathcal{M}$ *and associated data space* $\mathcal{A}$*. Consider an adversary* A *with access to encryption and decryption oracles* $\mathcal{E}_K$ *and* $\mathcal{D}_K$ *respectively, and an ideal primitive* π *and its inverse* π^{-1}*. We then define* A*'s detAuth advantage in attacking* Π *as*

$$Adv_{\Pi}^{detAuth}(A) = Pr[K \xleftarrow{\$} \mathcal{K} : A^{\mathcal{E}_K(\cdot,\cdot),\mathcal{D}_K(\cdot,\cdot),\pi,\pi^{-1}} \text{ suceeds in forgery}].$$

Definition 14 (PRF Advantage). *Let* $f : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ *be a function with a non-empty key space* $\mathcal{K}$ *and* A *a computationally bounded adversary with access to an oracle, where* $K \xleftarrow{\$} \mathcal{K}$ *and* $g \xleftarrow{\$} Func(\mathcal{X}, \mathcal{Y})$*. Thus, the PRF advantage of an adversary* A *on* f *is defined as*

$$Adv_f^{prf}(A) = \left| Pr[K \xleftarrow{\$} \mathcal{K} : A^{F(K,\cdot)} = 1] \right| - \left| Pr[g \xleftarrow{\$} Func(\mathcal{X}, \mathcal{Y}) : A^{g(\cdot)} = 1] \right|.$$

Definition 15 (rka-ind-advantage). *Let* Φ *be a set of mappings from* $\{0,1\}^n$ *to* $\{0,1\}^n$*. Let* A *be an adversary whose queries have the form* (ϕ, X)*, where* $X \in \{0,1\}^n$*,* $\phi \in \Phi$ *and* $\Phi \in \{\phi_{ident}, \phi_{const}\}$*. The rka-ind-advantage of* A *in a* Φ*-restricted-related key attack on* $\pi : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ *is given by*

$$\begin{aligned} Adv_{\pi(RK(\cdot,K),\cdot),\Phi}^{rka-ind}(A) \;=\; & \Big| Pr[K \xleftarrow{\$} \{0,1\}^n, A^{\pi(RK(\cdot,K),\cdot),\pi,\pi^{-1}} = 1] - \\ & Pr[g \xleftarrow{\$} Maps(\{0,1\}^{2n}, \{0,1\}^n), K \xleftarrow{\$} \{0,1\}^n, \\ & A^{g(RK(\cdot,K),\cdot),\pi,\pi^{-1}} = 1] \Big| \end{aligned}$$

where, on query (ϕ, X)*, the oracle* $\mathcal{O}(RK(\cdot, K), \cdot)$ *returns the value of* $\mathcal{O}(\phi(K), X)$ *to adversary* A*.*

9.2 Specifications of dAELM

In this work, we propose a new deterministic authenticated encryption scheme dAELM for memory constrained devices. In the next subsection, we will explain the operational environment for our scheme followed by the description of dEALM in Section 9.2.2.

9.2.1 An Operational Scenario For the Proposed Scheme

Our scenario assumes three entities—namely the sender, the receiver, and the cryptographic module—are interacting in the protocol. We assume that the encryption and decryption functionality is carried out inside the cryptographic module, which has limited storage capability and a built-in secret key. The purpose of using the cryptographic module is to provide a secure practical setup to prevent the leakage of the secret key from the device. The sender encrypts a message M, generates the ciphertext and tag pair (C, T), and sends the pair to the receiver. The receiver, having access to a similar cryptographic module, passes this (C, T) pair to the module for decryption. This cryptographic module decrypts ciphertext C to produce M, and without storing this M, he passes it directly to the MAC function for the calculation of tag T' (MAC computation is done in parallel with decryption operation). If at the end computed tag T' gets verified with the received tag, (i.e., $T = T'$) then cryptographic module reveals the session key K' to the receiver. After that, the receiver can decrypt C using K' to obtain the original M. The flow diagram for encryption and decryption are shown in Figure 9.1.

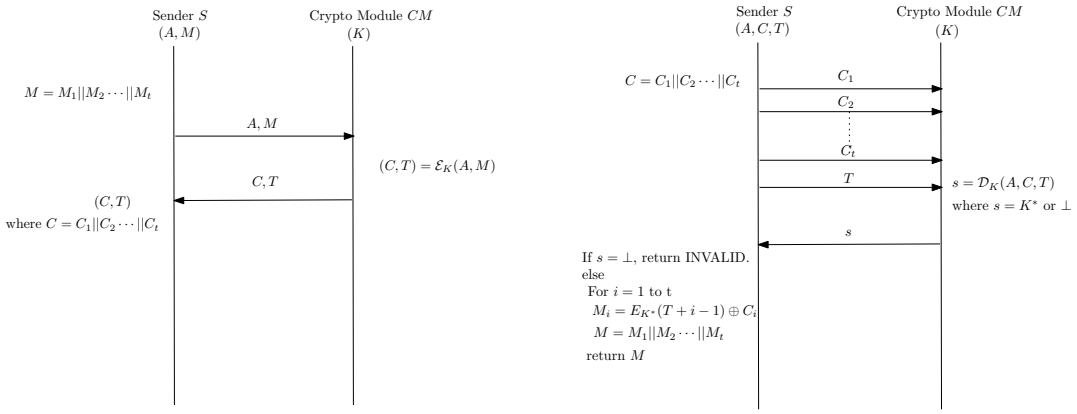


Figure 9.1: Flow diagram for encryption and decryption.

9.2.2 Description

This section defines the generic construction. The schematic structure of the proposed scheme is shown in Figure 9.2. Let $\mathcal{K}$ be a non-empty key space, and let $\mathcal{A} \subseteq \{0,1\}^*$ denote the associated data space. Encryption of dAELM takes three inputs key $K \in \mathcal{K}$, associated data $A \in \mathcal{A}$, and plaintext M, where associated data is optional and produces ciphertext tag pair (C, T) as output. Encryption works by first calculating MAC T over associated data A and plaintext M using key K. Further, T is encrypted using key $K + const$ to produce session key K^*. and plaintext M is then encrypted as C in a counter mode using T as a counter and K^* as key. Finally, (C, T) pair is produced as an output. For the decryption, first the session key K^* is generated using K and T, and ciphertext is then decrypted using the K^* to obtain plaintext M. Afterward, a tag is generated and compared with the received tag. If both tags are matched, then K^* is returned, otherwise invalid. Now using K^*, T, and C, original message M can be easily produced. Algorithms for encryption and decryption of dAELM constructions are shown in Algorithm 3 and 4 respectively, and the block diagrams representing encryption and decryption are shown in Figures 9.3 and 9.4, respectively.

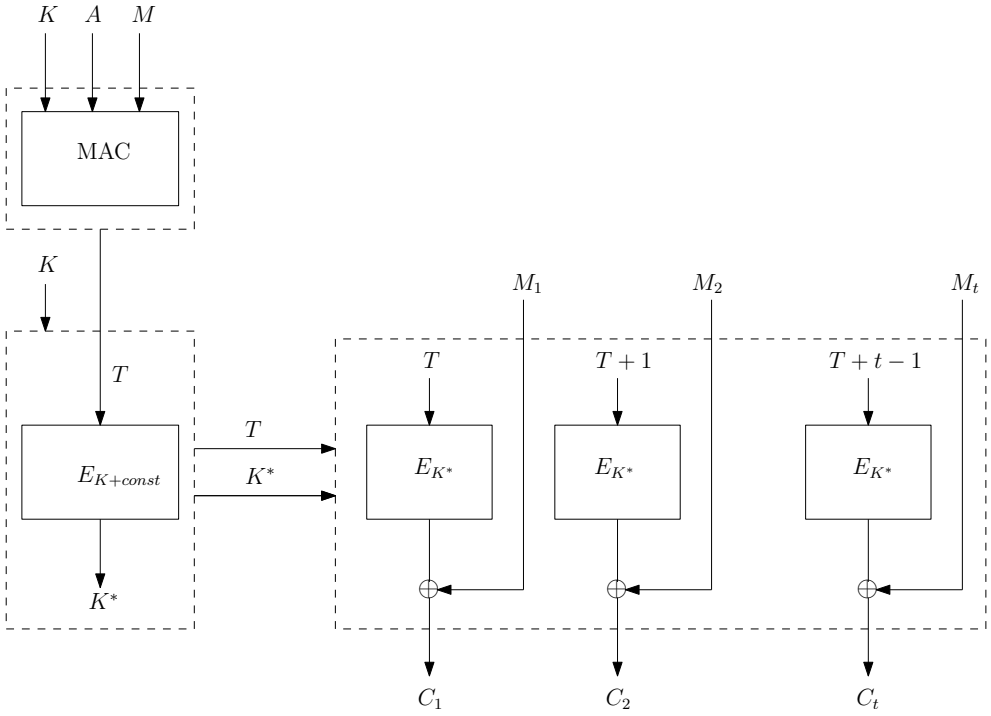


Figure 9.2: dAELM construction.

The underlying assumption for the dAELM construction is that it works for the

Algorithm 3: Encryption $\mathcal{E}_K(A, M)$

$M = M_1 \| \dots \| M_t$, where $|M_i| = n$ for $1 \leq i < t$ and $|M_t| \leq n$
$T = MAC(K, A, M)$
$K^* = E_{K+const}(T)$
for $i = 1 \to t-1$ **do**
    $y_i = E_{K^*}(T + i - 1)$
    $C_i = y_i \oplus M_i$
$y_i = E_{K^*}(T + t - 1)$ [First $|M_t|$ bits]
$C_n = y_i \oplus M_t$
$C = C_1 || C_2 \dots || C_t$
Return $(C||T)$

Algorithm 4: Decryption $\mathcal{D}_K\ (A, C, T)$

$K^* = E_{K+const}(T)$
$C = C_1 || C_2 \dots || C_t$ where $|C_i| = n$ for $1 < i < t$ and $|C_t| \leq n$
for $i = 1 \to t-1$ **do**
    $y_i = E_{K^*}(T + i - 1)$
    $M_i = y_i \oplus C_i$
$y_i = E_{K^*}(T + t - 1)$ [First $|C_t|$ bits]
$M_n = y_i \oplus C_t$
$M = M_1 || M_2 \dots || M_t$
$T' = MAC(K, A, M)$
if $T = T'$ **then**
    Return K^*
else
    Return $\perp$

block-cipher-based MAC only and works sequentially. In other words, we will consider only those MAC who processes message block-wise instead of processing as a whole.

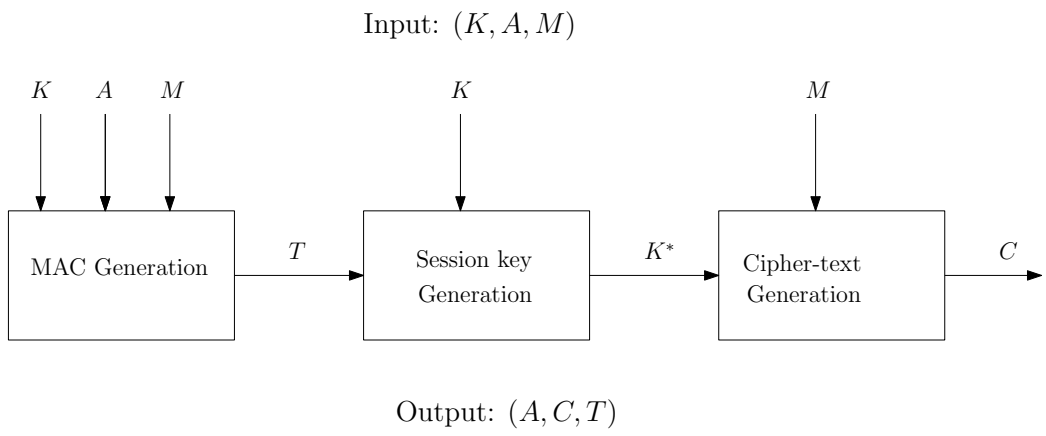


Figure 9.3: dAELM Encryption.

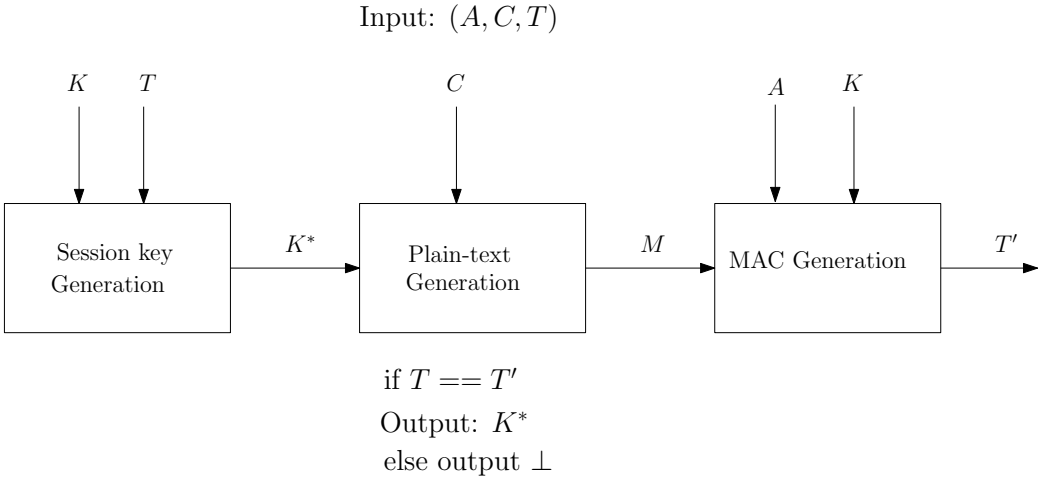


Figure 9.4: dEALM Decryption.

9.3 Security Results for the New Construction dAELM

In this section, we provide the security bounds for the privacy and authenticity of the dAELM construction. We gave the security proof in the ideal cipher model. For writing this proof, we have made an assumption that the underlying MAC in the proposed scheme works in a sequential manner, i.e., as soon as it receives a message block, MAC computation starts. We used a code-based game playing framework introduced by Bellare and Rogaway in [12] to write the proof. Theorem 3 states the privacy, and Theorem 4 states the authenticity of the proposed construction in the following sections.

9.3.1 Privacy

Theorem 3. *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the proposed DAE scheme with an ideal primitive π, that operates on n bits. The adversary A is given access to π and π^{-1}. Then the privacy advantage of adversary A is given by*

$$\begin{aligned} Adv_{\Pi}^{detPriv}(A) &= Pr[K \xleftarrow{\$} \mathcal{K} : A^{\mathcal{E}_K,\pi,\pi^{-1}} = 1] - Pr[K \xleftarrow{\$} \mathcal{K} : A^{\$(),\pi,\pi^{-1}} = 1] \\ Adv_{\Pi}^{detPriv}(A) &\leq Adv^{rka-ind}_{\pi(RK(\cdot,K),\cdot),\Phi,\pi}(\mathcal{B}_A) + \frac{\sigma(\sigma-1)}{2^{n+1}} + Adv^{prf}(D_A) \end{aligned}$$

where $\$$ is defined in Definition 2 in Section 9.1, and σ is the maximum number of block calls to π and π^{-1} by encryption $\mathcal{E}$ and decryption $\mathcal{D}$. $Adv^{prf}(D_A)$ is the prf advantage and $Adv^{rka-ind}_{\pi(RK(\cdot,K),\cdot),\Phi,\pi}(\mathcal{B}_A)$ is the related key advantage defined in Definitions 4 and 5 respectively in Section 9.1.

Proof. For the proof of privacy, we use code-based game-playing framework introduced by Bellare and Rogaway in [12]. We present a set of games from $G0$ to $G5$ (shown at the end

of this chapter in Fig. 9.6, 9.9, 9.8, 9.11 and 9.12). Starting from Game $G0$ and modifying it one step at a time, we construct a chain of games $G0 \longrightarrow G1 \longrightarrow \ldots \longrightarrow G5$. Here, Game $G0$ represents the actual construction, and Game $G5$ represents a completely random output. A description of each game is given the following:

G0: In this game, the encryption oracle perfectly simulates the proposed algorithm, and π, π^{-1} oracle simulates the ideal permutation and its inverse. As we will consider block-cipher-based MAC only, here we represent it by $MAC^{\pi(K,\cdot)}(M)$, where $\pi(K,\cdot)$ represents internal block-cipher calls on an input message M.

$$Pr[A^{\mathcal{E}_k,\pi,\pi^{-1}} = 1] = Pr[A^{G0} = 1].$$

G1: Game $G1$ is the same as $G0$ except in $G1$ the block cipher calls during MAC computation and session key generation has been replaced by subroutine g, which generates a random output for a given input. Adversary A cannot make a direct call to the subroutine g.

Let us assume that an adversary A distinguishes these two games. We build an adversary $\mathcal{B}$ against the subroutine g who uses adversary A. The process is as shown in appendix in Figure 9.7. □

Adversary $\mathcal{B}_A$: In this game, the challenger chooses the key K, and the adversary $\mathcal{B}_A$ is provided with the oracles $\mathcal{O}(RK(*,K),\cdot)$, which are simply a related key oracle and π oracle. For each (A, M), query adversary $\mathcal{B}$ internally calls oracle $\mathcal{O}$ and replies to adversary A with answer (C, T).

Finally, A receives the (C, T) pair for his queries either with the related key oracle or a random oracle. This game perfectly simulates the game $G0$ if $\mathcal{B}$ uses the related key oracle. Therefore, we have

$$Pr[G0] = Pr[\mathcal{B}_A^{\pi(RK(*,K),\cdot),\pi} = 1].$$

If adversary $\mathcal{B}$ uses a random oracle, then this game perfectly simulates game $G1$.

$$Pr[G1] = Pr[\mathcal{B}_A^{g(\cdot,\cdot),\pi} = 1].$$

Lemma 3. *For any $rka - ind$ adversary A with q queries, there exists an adversary $\mathcal{B}_A$ such that*

$$|Pr[A^{G1} = 1] - Pr[A^{G0} = 1]| \leq Adv^{rka-ind}_{\pi(RK(\cdot,K),\cdot),\Phi,\pi}(\mathcal{B}_A)$$

where $G0$ and $G1$ are shown in Figure 9.6, and $\mathcal{B}_A$ is defined in Figure 9.7. $\mathcal{B}_A$ can make only two queries: $(\phi_{ident},\cdot)$ and $(\phi_{const},\cdot)$. $\Phi = \{\phi_{ident}, \phi_{const}\}$, where $\phi_{ident}(x) = x$ and $\phi_{const}(x) = x \oplus const$.

G1': Game $G1'$ and $G1$ are exactly the same. Therefore,

$$Pr[A^{G1'} = 1] = Pr[A^{G1} = 1].$$

G2: Game $G1'$ and $G2$ differ only when a bad event occurs, otherwise, in absence of a bad event, both games are the same.

$$| Pr[A^{G1'} = 1] - Pr[A^{G2} = 1] | \leq Pr[bad].$$

A bad event occurs when an input to π and π^{-1} collide with the elements in set I_π. Therefore, the probability of a bad event will be equal to the probability of collision in π and π^{-1}. Hence, the probability difference between $G1$ and $G2$ is

$$| Pr[A^{G1'} = 1] - Pr[A^{G2} = 1] | \leq \frac{\sigma(\sigma - 1)}{2^{n+1}}.$$

G2': Games $G2'$ and $G2$ are exactly the same. Therefore,

$$Pr[A^{G2'} = 1] = Pr[A^{G2} = 1].$$

G3: Game $G2'$ and $G3$ are identical. In Game $G3$, the $\mathcal{E}$ oracle itself simulates the behavior of π, so it becomes independent of π. Hence, Games $G2$ and $G3$ are the same from an adversarial point of view.

$$Pr[A^{G3} = 1] = Pr[A^{G2'} = 1].$$

G4: Games $G3$ and $G4$ are the same except on Line 3.

Let us assume that an adversary A distinguishes these two games. We build an adversary D against the MAC oracle who uses adversary A. The process is as shown in appendix in Figure 9.10.

Adversary D_A: In this game, the challenger chooses the key K, and the adversary D_A is provided with the oracles $\mathcal{O}(\cdot)$, which is either a $MAC^{g(K,\cdot)}(\cdot)$ or $\$(\cdot)$ oracle. For each (A, M), query adversary D internally calls oracle $\mathcal{O}$ and replies to adversary A with answer (C, T).

Finally, A receives the (C, T) pair for his queries either with MAC oracle or random oracle. This game perfectly simulates Game $G3$ if D uses the MAC oracle. Therefore, we have

$$Pr[G3] = Pr[D_A^{MAC^{g(K,\cdot)}(\cdot),\pi,\pi^{-1}} = 1].$$

If adversary D uses a random oracle, then this game perfectly simulates Game $G4$.

$$Pr[G4] = Pr[D_A^{\$(\cdot),\pi,\pi^{-1}} = 1].$$

Lemma 4. *For any prf adversary A, there exists an adversary D_A such that*

$$|Pr[A^{G3} = 1] - Pr[A^{G4} = 1]| \leq Adv^{prf}(D_A).$$

G5: In Game $G4$, each ciphertext block is generated randomly and the concatenated ciphertext is returned to an adversary. Similarly, a tag is also generated randomly. However, in Game $G5$, ciphertext and tag of the desired length is selected as a random string and returned to an adversary. This move from $G4$ to $G5$ does not make any difference, as in both games output is generated as a completely random value. Therefore,

$$Pr[A^{G5} = 1] = Pr[A^{G4} = 1].$$

Finally, by using the fundamental lemma of the game-based framework,

$$\begin{aligned}
Adv_{\Pi}^{detPriv}(A) &= Pr[K \xleftarrow{\$} \mathcal{K} : A^{\mathcal{E}_K,\pi,\pi^{-1}} = 1] - Pr[K \xleftarrow{\$} \mathcal{K} : A^{\$_{pad}(),\pi,\pi^{-1}} = 1] \\
&= |\Pr[A^{G0} = 1] - Pr[A^{G5} = 1]| \\
&= |\,(Pr[A^{G0} = 1] - Pr[A^{G1} = 1])\,| + |\,(Pr[A^{G1} = 1] - Pr[A^{G1'} = 1])\,| \\
&\quad + |\,(Pr[A^{G1'} = 1] - Pr[A^{G2} = 1])\,| + |\,(Pr[A^{G2} = 1] - Pr[A^{G2'} = 1])\,| \\
&\quad + |\,(Pr[A^{G2'} = 1] - Pr[A^{G3} = 1])\,| + |\,(Pr[A^{G3} = 1] - Pr[A^{G4} = 1])\,| \\
&\quad + |\,(Pr[A^{G4} = 1] - Pr[A^{G5} = 1])\,| \\
&\leq Adv^{rka-ind}_{\pi(RK(\cdot,K),\cdot),\Phi,\pi}(\mathcal{B}_A) + 0 + \frac{\sigma(\sigma-1)}{2^{n+1}} + 0 + 0 + Adv^{prf}(D_A) + 0.
\end{aligned}$$

9.3.2 Authenticity

In this section, we analyze the security of the authenticity of the tag produced in our scheme. The forgery of an AE scheme is defined as the ability of an adversary A to generate a valid (A, M, C, T) tuple, without directly querying it to the encryption oracle. The adversary is allowed to make a limited number of queries to encryption, decryption, and the π and π^{-1} oracles. For an AE scheme, we say the adversary A is successful in forging if it outputs an (A, C, T) tuple, where $\mathcal{D}_K(A, C, T) \neq INVALID$ and adversary A has not asked fora query $\mathcal{E}_K(A, M)$ that resulted in response (C, T).

Let $Exp_{\Pi}^{Auth}(A)$ be the forging experiment for a given AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. Then the forging experiment is defined as follows:

1. Adversary A can query encryption oracle $\mathcal{E}$ and decryption oracle $\mathcal{D}$ at most $q_{enc} and\ q_{dec}$ times, respectively.

2. All the query responses of encryption oracle $\mathcal{E}$ are stored in a set, say, R. This set contains an (A, C, T) tuple, which has been returned by an encryption query.

3. If adversary A is able to generate a new (A, C, T) pair $\notin R$ that produces valid message M, then he wins and output is 1; otherwise, output is 0, after trying q_{dec} number of queries.

We say the adversary wins, i.e., he succeeds in creating a forgery if $Exp_{\Pi}^{Auth}(A) = 1$. Therefore, the advantage of an adversary in forging the scheme is defined as

$$Adv_{AE}^{detAuth}(A) = Pr[Exp_{\Pi}^{Auth}(A) = 1].$$

Suppose an adversary makes a q_{enc} number of encryption queries and stores a result in set R that consists of tuple (A_i, C_i, T_i), where $1 \leq i \leq q_{enc}$. Now his goal is to generate a new tuple $(A', C', T') \notin R$, which generates a valid M' on verification.

Theorem 4. *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the proposed authenticated encryption scheme with an ideal primitive (π) that operate on n bits. The adversary A is given access to encryption oracle $\mathcal{E}$, decryption oracle $\mathcal{D}$, and the π and π^{-1} oracles. Then $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is forgeable with the probability*

$$\begin{aligned} Pr[Exp_{\Pi}^{detAuth}(A) = 1] \quad \leq \quad & Adv^{rka-ind}_{\pi(RK(\cdot,K),\cdot),\Phi,\pi}(\mathcal{B}_A) + \frac{\sigma(\sigma-1)}{2^{n+1}} + Adv^{prf}(D_A) \\ & + \frac{q_{dec}^2}{2^n} + \frac{2}{2^n}. \end{aligned}$$

where σ is a maximum number of blocks to π, π^{-1} by encryption oracle $\mathcal{E}$, and decryption oracle $\mathcal{D}$ and q_{dec} is the number of queries to the decryption oracle. $Adv^{rka-ind}_{\pi(RK(\cdot,K),\cdot),\Phi,\pi}(\mathcal{B}_A)$ is the related key advantage, and $Adv^{prf}(D_A)$ is the prf advantage.

Proof. To give authenticity proof, we first use the same sequence of games ($G0$–$G4$) as in privacy proof with an additional decryption oracle. This sequence of games transforms the MAC into a pseudorandom function, and the block cipher calls into ideal permutation by considering their respective advantages. Here, we are directly using the advantages of games from the above privacy proof. After these transformations, we do further analysis. □

There are two possible cases to consider:

1. An adversary generates a forgery pair (A', C', T') which produces a valid M' such that $(*, *, T') \in R$; i.e., he chooses a new (A, C) pair, which corresponds to the same tag $T_i (= T')$ that has been previously used for some other (A_i, C_i), where $1 \leq i \leq q_{enc}$. T and T' will correspond to the same session key K^* for a given key K. This same K^* will result in the same output for further block cipher calls. This can be divided into cases.

   (a) $C' = C_i$: In this case, necessarily $A' \neq A_i$. As all block cipher outputs are the same, XORing them with $C (= C_i)$ will produce the same $M (= M_i)$. For a successful forgery, we need at least one parameter to be different in the (A', C', T') tuple. Since $C' = C_i$ and $T' = T_i$, A' should be different from A_i. Therefore, the probability of happening this case will be same as the birthday bound on the MAC output.

(b) $C' \neq C_i$: In this case, for $T_i(= T')$, $C' \neq C_i$ will result in $M' \neq M_i$. Therefore, the probability of this case will again become the same as the birthday bound on the MAC output.

Therefore, the overall probability of this case will be given by $\frac{q_{dec}^2}{2^n}$.

2. An adversary generates a forgery pair (A', C', T'), where T' is new, i.e. $(*, *, T') \notin R$. This case is further possible in two scenarios:

   (a) An adversary is able to guess the key K correctly; i.e. he has access to the system and can generate any number of valid queries. The probability of this case will be equivalent to guessing the key, which is $\frac{1}{2^n}$.

   (b) Suppose adversary guesses the tag T' correctly for a given key K and (A', M') pair. Then, for further calculations of ciphertext, he also needs to guess the session key K'. Then only he can generate the forgery. Therefore, the probability of happening this case will be the product of the probability of guessing the tag T' and session key K' that is equal to $\frac{1}{2^{2n}}$.

   Therefore, the overall probability of this case will be $\frac{1}{2^n}$.

Hence, the overall advantage of authenticity is given as

$$\begin{aligned} Adv_{\Pi}^{detAuth}(A) &= |\Pr[A^{G0} = 1] - Pr[A^{G4} = 1]| + \frac{q_{dec}^2}{2^n} + \frac{2}{2^n} \\ &= |\,(Pr[A^{G0} = 1] - Pr[A^{G1} = 1])\,| + |\,(Pr[A^{G1} = 1] - Pr[A^{G1'} = 1])\,| \\ &\quad + |\,(Pr[A^{G1'} = 1] - Pr[A^{G2} = 1])\,| + |\,(Pr[A^{G2} = 1] - Pr[A^{G2'} = 1])\,| \\ &\quad + |\,(Pr[A^{G2'} = 1] - Pr[A^{G3} = 1])\,| + |\,(Pr[A^{G3} = 1] - Pr[A^{G4} = 1])\,| \\ &\quad + \frac{q_{dec}^2}{2^n} + \frac{2}{2^n}. \end{aligned}$$

$$\begin{aligned} Adv_{\Pi}^{detAuth}(A) &\leq Adv_{\pi(RK(\cdot,K),\cdot),\Phi,\pi}^{rka-ind}(\mathcal{B}_A) + 0 + \frac{\sigma(\sigma-1)}{2^{n+1}} + 0 + 0 + Adv^{prf}(D_A) \\ &\quad + \frac{q_{dec}^2}{2^n} + \frac{2}{2^n}. \end{aligned}$$

9.4 Comparison

In Table 9.1, we present a comparison of the newly proposed scheme dAELM with existing schemes sp-AELM [3], OAE2 [45], and SIV [68]. Among these four schemes, sp-AELM and OAE2 are randomized schemes requiring nonces to produce the randomness. The remaining

Parameters	**sp-AELM** [3]	**OAE2** [45]	**SIV** [68]	**dAELM** [This work]
Type of AE	Randomized	Randomized	Deterministic	Deterministic
Online Encryption	Yes	Yes	No	No
Online Decryption	No	Yes	No	No
Low memory support	Yes	Yes	No	Yes
Misuse-resistant	No	No	Yes	yes
Segmentation	Fixed size block	Variable size segment	Fixed size block	Fixed size block
Ciphertext expansion	τ bits	τ bits per segment	τ bits	τ bits

Table 9.1: Comparison of dEALM with sp-AELM, OAE2 and SIV: where τ represents the tag length

two schemes, the SIV and the dAELM are deterministic. They use the available message entropy to omit the overhead of a nonce. Schemes sp-AELM and OAE2 support online encryption, whereas SIV and dAELM do not provide online encryption features due to their misuse-resistant behavior. As mentioned in [45], misuse-resistant AE schemes cannot be online since every bit of the ciphertext must depend on every bit of plaintext in a misuse-resistant scheme. This ensures that one cannot output the first bit of a ciphertext before reading the last bit of the plaintext. Among the four schemes, only OAE2 supports online decryption at the cost of generating a tag for every individual segment of the plaintext. All these schemes provide low memory support, except SIV. Further, sp-AELM and OAE2 are vulnerable to a nonce-repeating attack, while this is not the case with SIV and dAELM.

OAE2 operates by dividing the plaintext into variable size segments of user-determined lengths, whereas sp-AELM, SIV, and dAELM process the plaintext into fixed size blocks of fixed lengths. In sp-AELM, SIV, and dAELM, the ciphertext length is a sum of plaintext and tag length ($\tau bits$), whereas in the OAE2 scheme the tag is generated for each segment.

9.5 Software Implementation

We implemented the proposed scheme dAELM in ATmega128 microcontroller and compared the performance against an existing SIV scheme. ATmega128 microcontroller is an 8-bit AVR RISC-based microcontroller, which combines 128 KB of programmable flash memory, 4 KB SRAM, and a 4 KB EEPROM. The device supports throughput of 16 MIPS at 16 MHz and operates at 4.5–5.5 volts. We used the C language for programming ATmega 128. For the implementation of dAELM and SIV, we used the AES-128 as an underlying block cipher and HMACSHA1 as an underlying MAC function. We executed the experiments for

the messages of various length and analyzed how much memory is taken by the program performing encryption and decryption.

Pseudocode of decryption is shown in Algorithms 5 and 6. Three additional functions are used for computing the HMACSHA1 tag without describing the details due to its wide compatibility. HMACSHA1.INIT() initializes an HMACSHA1 structure to use the hash function and the key. HMACSHA1.UPDATE() is repeatedly called with chunks of the messages to be authenticated. HMACSHA1.FINAL() completes the HMACSHA1 operation after the last message is processed and returns the message authentication code. For the decryption process, instead of storing the entire decrypted message after decrypting the entire ciphertext, it decrypts the four blocks (each of 128 bit), and HMACSHA1.UPDATE() takes them as an input repeatedly.

Algorithm 5: Decryption $\mathcal{D}_K(A, C, T)$

$C = C_1||C_2 \ldots ||C_t$ where $|C_i| = 512$ for $1 \leq i < t$ and $|C_t| \leq 512$
$K^* = AES_{K \oplus const}(T)$
$CTR = T$
HMACSHA1.INIT(K)
for $i = 1 \rightarrow t - 1$ **do**
    $M_i = AES\_CTR\_DEC\_4BLOCKS(K^*, C_i, CTR + 4(i-1))$
    HMACSHA1.UPDATE(M_i)
$M_t = AES\_CTR\_DEC\_4BLOCKS(K^*, C_t, CTR + 4(i-1))$ [First $|C_t|$ bits]
HMACSHA1.UPDATE(M_t)
$T' =$ HMACSHA1.FINAL()
if $T = T'$ **then**
    Return K^*
else
    Return $\perp$

Algorithm 6: $AES\_CTR\_DEC\_4BLOCKS(K, C, CTR)$

$C = C_1||C_2||C_3||C_4$ where $|C_i| = 128$ for $i = 1, 2, 3, 4$
$M_1 = C_1 \oplus AES_K(CTR)$
$M_2 = C_2 \oplus AES_K(CTR + 1)$
$M_3 = C_3 \oplus AES_K(CTR + 2)$
$M_4 = C_4 \oplus AES_K(CTR + 3)$
$M = M_1||M_2||M_3||M_4$
Return M

Figure 9.5 shows the memory usage during the decryption process for the various messages lengths. For the messages of length less than 512 bits, memory required by the decryption process for dAELM is a bit more than SIV due to one extra AES call for session key generation. This difference in memory usage of only 512 bits is because of the

consideration of HMAC-SHA1 in our implementation, which process a message in 512 bit blocks. After 512 bits, memory usage becomes constant for dAELM, regardless of message length, whereas in SIV, the memory usage depends on the length of a message.

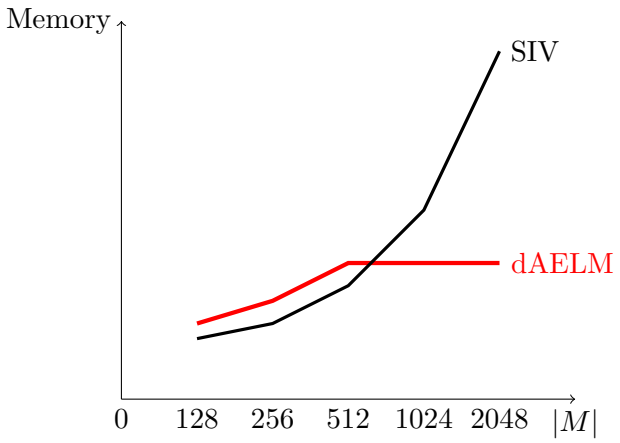


Figure 9.5: A graph showing memory usage by the cryptomodule during the decryption process, where the x-axis represents the length of message and the y-axis represents the memory usage in bytes. This is the case when we used HMAC-SHA1 as a MAC function, which processes the message in 512 bit blocks.

In general, the memory required by dAELM is $|K| + |K^*| + |A| + |B|$, where $|K|$ and $|K^*|$ is a length of a key and a session key, respectively, $|A|$ is the length of an associated data, and $|B|$ is the block size of the message processed in the MAC. For example, $|B|$ is 512 bits in the case of HMAC-SHA1. However, the memory required by SIV is $|K| + |A| + |M|$, where $|K|, |A|$, and $|M|$ represent the length of a key, associated data, and a message, respectively. In other words, memory usage for dAELM depends on the block size of MAC, which is constant, whereas, for SIV, it depends on the message length. If we consider a MAC with a small block size, then dAELM will perform better than SIV, even for short messages.

9.6 Discussion

In this work, our proposed *AE* scheme targets the devices, which have a limited amount of memory and are incapable of processing the large message. These memory-constrained devices include, in tiny IoT devices, cryptomodules such as TPMs (trusted platform modules), which are used in laptops, tablets, smartphones, set-top-boxs, ATM machines, etc. A cryptomodule is basically a secure cryptoprocessor used for carrying out the specific cryptographic operations and storing secret keys. These cryptomodules usually have a storage capacity of only a few bytes, which may not be sufficient to process long messages.

For example, ATMELAT97SC3204 is a TPM that has only 1732 bytes of storage for the user-defined data. Similarly, these small IoT devices also have a limited amount of storage. Under these memory constraints, it becomes impossible to process a large data in one go. Hence, our proposed scheme provides a solution for all those memory-constrained devices to process a long message.

9.7 Summary

In this chapter, we presented a new DAE scheme (dAELM) that is suitable for memory-constrained scenarios. Our proposed construction is based on the SIV mode and does not require nonce as an input. We also provide the security bounds for both the privacy and the authenticity of the construction.

Appendix

Here, we provide the games used in the privacy and authenticity proofs of dAELM.

Games for Privacy and Authenticity Proofs for dAELM

We make sequence of games used in security proofs of Theorem 1 and 2. We mention priv and Auth in each game to include or not to include particular type of queries while writing the proofs.

Game $\boxed{G0}$ **and** $\boxed{\textbf{G1}}$:Initialize $K \xleftarrow{\$} \{0,1\}^n$, $I_\pi = \phi$, $I_T = \phi$

priv Auth

On Encryption-Query $(A, M, flag)$

1. Parse $M = M_1||M_2 \ldots ||M_t$ where $|M_i| = n$ for $1 \leq i \leq t-1$ and $|M_t| \leq n$
2. $M' \leftarrow Pad_{out}(A, M)$
3. $\boxed{T \leftarrow MAC^{\pi(K,\cdot)}(M'),\ I_T = I_T \cup \{(T, K, M')\}}$
   $T \leftarrow MAC^{g(K,\cdot)}(M')$
   if $\exists\ T$ s.t. $(T, *, *) \in I_T$ then $bad \rightarrow true$
4. $\boxed{K^* \leftarrow \pi(K \oplus const, T)}$
   $K^* \leftarrow g(K \oplus const, T)$
5. **for** $i = 1 \rightarrow t-1$ **do**
   $y_i \leftarrow \pi(K^*, T+i-1)$
   $C_i \leftarrow y_i \oplus M_i$
6. $y_t \leftarrow \pi((K^*, T+t-1))$ [First $|M_t|$ bits]
7. $C_t \leftarrow y_t \oplus M_t$
8. $C = C_1||C_2 \ldots ||C_t$
9. Return (A, C, T)

Auth

On Decryption-Query (A, C, T)

1. Parse $C = C_1||C_2 \ldots ||C_t$ where $|C_i| = n$ for $1 \leq i \leq t-1$ and $|C_t| \leq n$
2. $\boxed{K^* \leftarrow \pi(K \oplus const, T)}$
   $K^* \leftarrow g(K \oplus const, T)$
3. **for** $i = 1 \rightarrow t-1$ **do**
   $y_i \leftarrow \pi(K^*, T+i-1)$
   $M_i \leftarrow y_i \oplus C_i$
4. $y_t \leftarrow \pi(K^*, T+t-1)$ [First $|M_t|$ bits]
5. $M_t \leftarrow y_t \oplus C_t$
6. $M = M_1||M_2 \ldots ||M_t$
7. $M' \leftarrow Pad_{out}(A, M)$
8. $\boxed{T' \leftarrow MAC^{\pi(K,\cdot)}(M'),\ I_T = I_T \cup \{(T, K, M')\}}$
   $T' \leftarrow MAC^{g(K,\cdot)}(M')$
   if $\exists\ T'$ s.t. $(T', *, *) \in I_T$ then $bad \rightarrow true$
9. **if** $(T == T')$ **then**
   Return K^*
   else
   Return $\perp$

on $Pad_{out}(A, M)$

1. $M' = bin_{64}(|A|)||A||M$
2. return M'

priv Auth

On π **query** (K', m)

1. if $\exists\ v$ s.t. $(K', m, v) \in I_\pi$ then return v
2. $v \xleftarrow{\$} \{0,1\}^n$
3. if $\exists * $ s.t. $(K', *, v) \in I_\pi$, then $v \xleftarrow{\$} \{0,1\}^n \setminus \{v : (K', *, v) \in I_\pi\}$, where $* \in \{0,1\}^n$
4. $I_\pi = I_\pi \bigcup \{(K', m, v)\}$
5. return v;

priv Auth

On π^{-1} **query** (K', v)

1. if $\exists\ m$ s.t. $(K', m, v) \in I_\pi$ then return m
2. $m \xleftarrow{\$} \{0,1\}^n$
3. if $\exists * $ s.t. $(K', m, *) \in I_\pi$, then $M \xleftarrow{\$} \{0,1\}^n \setminus \{M : (K', m, *) \in I_\pi\}$, where $* \in \{0,1\}^n$
4. $I_\pi = I_\pi \bigcup \{(K', m, v)\}$
5. return m;

Subroutine $g(K', x)$

1. if $\exists\ y$ s.t. $(K', x, y) \in I_g$ then return y
2. $y \leftarrow \{0,1\}^n$
3. $I_g = I_g \bigcup \{(K', x, y)\}$
4. return y;

Figure 9.6: Game G0 and G1

Adversary Game $\mathcal{B}_A^{\mathcal{O}(RK(*,K),\cdot),\pi,\pi^{-1}}$,$K \xleftarrow{\$} \{0,1\}^n$ (chosen by challenger) where $\mathcal{O}$ is either $\pi(RK(*,K),\cdot)$ or $g(RK(*,K),\cdot)$, $* \in \{\phi_{const}, \phi_{ident}\}$ and $g \xleftarrow{\$} Maps(\{0,1\}^{2n},\{0,1\}^n)$.

priv Auth

On `Sim`-Encryption-Query $(A, M, flag)$

1. Parse $M = M_1||M_2 \dots ||M_t$ where $|M_i| = n$ for $1 \le i \le t-1$ and $|M_t| \le n$
2. $M' \leftarrow Pad_{out}(A, M)$
3. $T \leftarrow MAC^{\mathcal{O}(RK(\phi_{ident},K),\cdot)}(M')$, $I_T = I_T \cup \{(T, K, M')\}$
4. $K^* \leftarrow \mathcal{O}(RK(\phi_{const}, K), T)$
5. **for** $i = 1 \to t-1$ **do**
   $y_i \leftarrow \texttt{Sim} - \pi(K^*, T+i-1)$
   $C_i \leftarrow y_i \oplus M_i$
6. $y_t \leftarrow \texttt{Sim} - \pi(K^*, T+t-1)$ [First $|M_t|$ bits]
7. $C_t \leftarrow y_t \oplus M_t$
8. $C = C_1||C_2 \dots ||C_t$
9. Return (A, C, T)

on $Pad_{out}(A, M)$

1. $M' = bin_{64}(|A|)||A||M$
2. return M'

priv Auth

on $\texttt{Sim} - \pi$ **query** (K', m)

1. $v = \pi(K', m)$
2. Return v

Auth

On `Sim`-Decryption-Query (A, C, T)

1. Parse $C = C_1||C_2 \dots ||C_t$ where $|C_i| = n$ for $1 \le i \le t-1$ and $|C_t| \le n$
2. $K^* \leftarrow \mathcal{O}(RK(\phi_{const}, K), T)$
3. **for** $i = 1 \to t-1$ **do**
   $y_i \leftarrow \texttt{Sim} - \pi(K^*, T+i-1)$
   $M_i \leftarrow y_i \oplus C_i$
4. $y_t \leftarrow \texttt{Sim} - \pi(K^*, T+t-1)$ [First $|M_t|$ bits]
5. $M_t \leftarrow y_t \oplus C_t$
6. $M = M_1||M_2 \dots ||M_t$
7. $M' \leftarrow Pad_{out}(A, M)$
8. $T' \leftarrow MAC^{\mathcal{O}(RK(\phi_{ident},K),\cdot)}(M')$, $I_T = I_T \cup \{(T', K, M')\}$
9. **if** $(T == T')$ **then**
   Return K^*
   else
   Return $\perp$

priv Auth

on $\texttt{Sim} - \pi^{-1}$ **query** (K', v)

1. $m = \pi^{-1}(K', v)$
2. Return m

Figure 9.7: Adversary $\mathcal{B}_A$

Game $\boxed{G1'}$ **and G2:** Initialize $K \xleftarrow{\$} \{0,1\}^n$, $I_\pi = \phi$,

priv Auth

On Encryption-Query $(A, M, flag)$

1. Parse $M = M_1||M_2 \ldots||M_t$ where $|M_i| = n$ for $1 \le i \le t-1$ and $|M_t| \le n$
2. $M' \leftarrow Pad_{out}(A, M)$
3. $T \leftarrow MAC^{g(K,\cdot)}(M')$
4. $K^* \leftarrow g(K \oplus const, T)$
5. **for** $i = 1 \to t-1$ **do**
   $y_i \leftarrow \pi(K^*, T+i-1)$
   $C_i \leftarrow y_i \oplus M_i$
6. $y_t \leftarrow \pi(K^*, T+t-1)$ [First $|M_t|$ bits]
7. $C_t \leftarrow y_t \oplus M_t$
8. $C = C_1||C_2 \ldots||C_t$
9. Return (A, C, T)

Auth

On Decryption-Query (A, C, T)

1. Parse $C = C_1||C_2 \ldots||C_t$ where $|C_i| = n$ for $1 \le i \le t-1$ and $|C_t| \le n$
2. $K^* \leftarrow g(K \oplus const, T)$
3. **for** $i = 1 \to t-1$ **do**
   $y_i \leftarrow \pi(K^*, T+i-1)$
   $M_i = y_i \oplus C_i$
4. $y_t \leftarrow \pi(K^*, T+t-1)$ [First $|M_t|$ bits]
5. $M_t \leftarrow y_t \oplus C_t$
6. $M = M_1||M_2 \ldots||M_t$
7. $M' \leftarrow Pad_{out}(A, M)$
8. $T' \leftarrow MAC^{g(K,\cdot)}(M')$
9. **if** $(T == T')$ **then**
   Return K^*
   else
   Return $\perp$

on $Pad_{out}(A, M)$

1. $M' = bin_{64}(|A|)||A||M$
2. return M'

priv Auth

On π **query** (K', m)

1. if $\exists\ v$ s.t. $(K', m, v) \in I_\pi$ then return v
2. $v \xleftarrow{\$} \{0,1\}^n$
3. if $\exists * $ s.t. $(K', *, v) \in I_\pi$, then **bad** $\leftarrow$ **true** and $\boxed{v \xleftarrow{\$} \{0,1\}^n \setminus \{v : (K', *, v) \in I_\pi\},}$ where $* \in \{0,1\}^n$
4. $I_\pi = I_\pi \bigcup \{(K', m, v)\}$
5. return v;

priv Auth

On π^{-1} **query** (K', v)

1. if $\exists\ m$ s.t. $(K', m, v) \in I_\pi$ then return m
2. $m \xleftarrow{\$} \{0,1\}^n$
3. if $\exists * $ s.t.$(K', m, *) \in I_\pi$, then **bad** $\leftarrow$ **true** and $\boxed{m \xleftarrow{\$} \{0,1\}^n \setminus \{m : (K', m, *) \in I_\pi\},}$ where $* \in \{0,1\}^n$
4. $I_\pi = I_\pi \bigcup \{(K', m, v)\}$
5. return m;

Subroutine $g(K', x)$

1. if $\exists\ y$ s.t. $(K', x, y) \in I_g$ then return y
2. $y \leftarrow \{0,1\}^n$
3. $I_g = I_g \bigcup \{(K', x, y)\}$
4. return y;

Figure 9.8: Game G1' and G2

Game $\boxed{G2'}$ **and** $\boxed{\textbf{G3}}$**:** Initialize $K \xleftarrow{\$} \{0,1\}^n$, $I_\pi = \phi$,

priv Auth

On Encryption-Query $(A, M, flag)$

1. Parse $M = M_1||M_2 \ldots ||M_t$ where $|M_i| = n$ for $1 \leq i \leq t-1$ and $|M_t| \leq n$
2. $M' \leftarrow Pad_{out}(A, M)$
3. $T \leftarrow MAC^{g(K,\cdot)}(M')$
4. $K^* \xleftarrow{\$} \{0,1\}^n$
5. **for** $i = 1 \rightarrow t-1$ **do**
   - $\boxed{y_i \leftarrow \pi(K^*, T+i-1)}$
   - if $\exists\ v$ s.t. $(K', T+i-1, v) \in I_\pi$ then $y_i \leftarrow v$
   - else $y_i \xleftarrow{\$} \{0,1\}^n$, $I_\pi = I_\pi \cup \{K', T+i-i, y_i\}$
   - $C_i \leftarrow y_i \oplus M_i$
6. $\boxed{y_t \leftarrow \pi(K^*, T+t-1)}$
   - if $\exists\ v$ s.t. $(K', T+t-1, v) \in I_\pi$ then $y_t \leftarrow v$
   - else $y_t \xleftarrow{\$} \{0,1\}^n$, $I_\pi = I_\pi \cup \{(K', T+t-i, y_i)\}$
7. $y_t \leftarrow y_t$ [first $|M_t|$ bits]
8. $C_t \leftarrow y_t \oplus M_t$
9. $C = C_1||C_2 \ldots ||C_t$
10. Return (A, C, T)

Auth

On Decryption-Query (A, C, T)

1. Parse $C = C_1||C_2 \ldots ||C_t$ where $|C_i| = n$ for $1 \leq i \leq t-1$ and $|C_t| \leq n$
2. $K^* \xleftarrow{\$} \{0,1\}^n$
3. **for** $i = 1 \rightarrow t-1$ **do**
   - $\boxed{y_i \leftarrow \pi(K^*, T+i-1)}$
   - if $\exists\ v$ s.t. $(K', m, v) \in I_\pi$ then $y_i \leftarrow v$
   - else $y_i \xleftarrow{\$} \{0,1\}^n$, $I_\pi = I_\pi \cup \{K', T+i-i, y_i\}$
   - $M_i = y_i \oplus C_i$
4. $\boxed{y_t \leftarrow \pi(K^*, T+t-1)}$
   - if $\exists\ v$ s.t. $(K', T+t-1, v) \in I_\pi$ then $y_t \leftarrow v$
   - else $y_t \xleftarrow{\$} \{0,1\}^n$, $I_\pi = I_\pi \cup \{(K', T+t-i, y_i)\}$
5. $y_t \leftarrow y_t$ [first $|C_t|$ bits]
6. $M_t \leftarrow y_t \oplus C_t$
7. $M = M_1||M_2 \ldots ||M_t$
8. $M' \leftarrow Pad_{out}(A, M)$
9. $T' \leftarrow MAC^{g(K,\cdot)}(M')$
10. **if** $(T == T')$ **then**
    - Return K^*

    else
    - Return $\perp$

on $Pad_{out}(A, M)$

1. $M' = bin_{64}(|A|)||A||M$
2. return M'

priv Auth

On π **query** (K', m)

1. if $\exists\ v$ s.t. $(K', m, v) \in I_\pi$ then return v
2. $v \xleftarrow{\$} \{0,1\}^n$
3. $I_\pi = I_\pi \bigcup \{(K', m, v)\}$
4. return v;

priv Auth

On π^{-1} **query** (K', v)

1. if $\exists\ m$ s.t. $(K', m, v) \in I_\pi$ then return m
2. $m \xleftarrow{\$} \{0,1\}^n$
3. $I_\pi = I_\pi \bigcup \{(K', m, v)\}$
4. return m;

Subroutine $g(K', x)$

1. if $\exists\ y$ s.t. $(K', x, y) \in I_g$ then return y
2. $y \leftarrow \{0,1\}^n$
3. $I_g = I_g \bigcup \{(K', x, y)\}$
4. return y;

Figure 9.9: Game G2' and G3

Adversary Game $D_A^{\mathcal{O}}$ where $\mathcal{O}$ is either $MAC^{g(K,\cdot)}$ or \$.

priv Auth

On `Sim`-Encryption-Query $(A, M, flag)$

1. Parse $M = M_1||M_2 \ldots ||M_t$ where $|M_i| = n$ for $1 \leq i \leq t-1$ and $|M_t| \leq n$
2. $T \leftarrow \mathcal{O}(M')$
3. $K^* \xleftarrow{\$} \{0,1\}^n$
4. **for** $i = 1 \rightarrow t-1$ **do**
   - $y_i \xleftarrow{\$} \{0,1\}^n$
   - $I'_\pi = I'_\pi \cup \{K', T+i-i, y_i\}$
   - $C_i \leftarrow y_i \oplus M_i$
5. $y_t \xleftarrow{\$} \{0,1\}^n$ [First $|M_t|$ bits]
6. $I'_\pi = I'_\pi \cup \{(K', T+t-i, y_i)\}$
7. $y_t \leftarrow y_t$ [first $|M_t|$ bits]
8. $C_t \leftarrow y_t \oplus M_t$
9. $C = C_1||C_2 \ldots ||C_t$
10. Return (A, C, T)

on $Pad_{out}(A, M)$

1. $M' = bin_{64}(|A|)||A||M$
2. return M'

priv Auth

On `Sim`-π query (K', m)

1. $v = \pi(K', m)$
2. return v;

Auth

On `Sim`-Decryption-Query (A, C, T)

1. Parse $C = C_1||C_2 \ldots ||C_t$ where $|C_i| = n$ for $1 \leq i \leq t-1$ and $|C_t| \leq n$
2. $K^* \xleftarrow{\$} \{0,1\}^n$
3. **for** $i = 1 \rightarrow t-1$ **do**
   - **if** $(K', T+i-1, y) \in I'_\pi$ **then**
     - $y_i \leftarrow y$
   - **else**
     - $y_i \xleftarrow{\$} \{0,1\}^n$
     - $I'_\pi = I'_\pi \cup \{(K', T+i-i, y_i)\}$
   - $M_i = y_i \oplus C_i$
4. **if** $(K', T+t-1, y) \in I'_\pi$ **then**
   - $y_t \leftarrow y$
   - **else**
     - $y_t \xleftarrow{\$} \{0,1\}^n$
     - $I'_\pi = I'_\pi \cup \{K', T+t-i, y_t\}$
5. $y_t \leftarrow y_t$ [first $|C_t|$ bits]
6. $M_t \leftarrow y_t \oplus C_t$
7. $M = M_1||M_2 \ldots ||M_t$
8. $T' \xleftarrow{\$} \{0,1\}^n$
9. **if** $(T == T')$ **then**
   - Return K^*
   - **else**
     - Return $\perp$

priv Auth

On `Sim`$-\pi^{-1}$ query (K', v)

1. $m = \pi^{-1}(K', v)$
2. return M;

Figure 9.10: Adversary D_A

Game G4: Initialize $K \xleftarrow{\$} \{0,1\}^n$

priv Auth

On Encryption-Query $(A, M, flag)$

1. Parse $M = M_1||M_2 \ldots ||M_t$ where $|M_i| = n$ for $1 \leq i \leq t-1$ and $|M_t| \leq n$
2. $M' \leftarrow Pad_{out}(A, M)$
3. $T \xleftarrow{\$} \{0,1\}^n$
4. $K^* \xleftarrow{\$} \{0,1\}^n$
5. **for** $i = 1 \rightarrow t-1$ **do**
   - $y_i \xleftarrow{\$} \{0,1\}^n$
   - $C_i \leftarrow y_i \oplus M_i$
6. $y_t \xleftarrow{\$} \{0,1\}^n$ [First $|M_t|$ bits]
7. $C_t \leftarrow y_t \oplus M_t$
8. $C = C_1||C_2 \ldots ||C_t$
9. Return (A, C, T)

on $Pad_{out}(A, M)$

1. $M' = bin_{64}(|A|)||A||M$
2. return M'

priv Auth

On π **query** (K', M)

1. if $\exists\ v$ s.t. $(K', M, v) \in I_\pi$ then return v
2. $v \xleftarrow{\$} \{0,1\}^n$
3. $I_\pi = I_\pi \bigcup \{(K', M, v)\}$
4. return v;

Auth

On Decryption-Query (A, C, T)

1. Parse $C = C_1||C_2 \ldots ||C_t$ where $|C_i| = n$ for $1 \leq i \leq t-1$ and $|C_t| \leq n$
2. $K^* \xleftarrow{\$} \{0,1\}^n$
3. **for** $i = 1 \rightarrow t-1$ **do**
   - $y_i \xleftarrow{\$} \{0,1\}^n$
   - $M_i = y_i \oplus C_i$
4. $y_t \xleftarrow{\$} \{0,1\}^n$ [First $|M_t|$ bits]
5. $M_t \leftarrow y_t \oplus C_t$
6. $M = M_1||M_2 \ldots ||M_t$
7. $M' \leftarrow Pad_{out}(A, M)$
8. $T' \xleftarrow{\$} \{0,1\}^n$
9. **if** $(T == T')$ **then**
   - Return K^*

   else
   - Return $\perp$

priv Auth

On π^{-1} **query** (K', v)

1. if $\exists\ M$ s.t. $(K', M, v) \in I_\pi$ then return M
2. $M \xleftarrow{\$} \{0,1\}^n$
3. $I_\pi = I_\pi \bigcup \{(K', M, v)\}$
4. return M;

Figure 9.11: Game $G4$

Game G5: Initialize $K \xleftarrow{\$} \{0,1\}^n$

priv Auth

On Encryption-Query $(A, M, flag)$

1. Parse $M = M_1||M_2 \ldots ||M_t$ where $|M_i| = n$ for $1 \leq i \leq t-1$ and $|M_t| \leq n$
2. $T \xleftarrow{\$} \{0,1\}^n$
3. $K^* \xleftarrow{\$} \{0,1\}^n$
4. **for** $i = 1 \rightarrow t$ **do**
   $C_i \xleftarrow{\$} \{0,1\}^n$
5. $C = C_1||C_2 \ldots ||C_t$
6. Return (A, C, T)

on $Pad_{out}(A, M)$

1. $M' = bin_{64}(|A|)||A||M$
2. return M'

priv Auth

On π **query** (K', M)

1. if $\exists\ v$ s.t. $(K', M, v) \in I_\pi$ then return v
2. $v \xleftarrow{\$} \{0,1\}^n$
3. $I_\pi = I_\pi \bigcup \{(K', M, v)\}$
4. return v;

Auth

On Decryption-Query (A, C, T)

1. Parse $C = C_1||C_2 \ldots ||C_t$ where $|C_i| = n$ for $1 \leq i \leq t-1$ and $|C_t| \leq n$
2. $K^* \xleftarrow{\$} \{0,1\}^n$
3. **for** $i = 1 \rightarrow t-1$ **do**
   $M_i = \xleftarrow{\$} \{0,1\}^n$
4. $M = M_1||M_2 \ldots ||M_t$
5. $T' \xleftarrow{\$} \{0,1\}^n$
6. **if** $(T == T')$ **then**
   Return K^*

   else
   Return $\perp$

priv Auth

On π^{-1} **query** (K', v)

1. if $\exists\ M$ s.t. $(K', M, v) \in I_\pi$ then return M
2. $M \xleftarrow{\$} \{0,1\}^n$
3. $I_\pi = I_\pi \bigcup \{(K', M, v)\}$
4. return M;

Figure 9.12: Game G5

Chapter 10

RSAEB: Modified SAEB Secure in RUP Settings

Authenticated encryption (AE) combines privacy with data integrity, and in the process of decryption, the plaintext is always kept until successful verification. But in applications with insufficient memory or with realtime requirement, release of unverified plaintext is unavoidable. Furthermore most of present online AE schemes claim to keep the unverified plaintext, leading to online encryption but offline decryption, which seems unreasonable for online applications. Thus, security of the releasing unverified plaintext (RUP) setting, especially for online AE scheme need to be taken seriously. The notion of plaintext awareness (PA) together with IND-CPA have been formalized to achieve privacy in RUP setting by Andreeva et al. in 2014. In this work we have modified an existing lightweight authenticated encryption scheme called SAEB, to make it secure in RUP (Releasing Unverified Plaintext) settings. SAEB is a lightweight block-cipher based AEAD mode of operation designed by Naito et. al in 2018.

SAEB [60] is a recently introduced lightweight block cipher based AEAD mode of operation designed by Naito et. al. It follows the sponge based design methodology, the technique conventionally used with permutation is applied to block-cipher. It is an online, inverse-free, XOR only AE scheme and also requires the minimum state size, same as block size of block-cipher. It also provide support for handling static associated data. SAEB provides integrity in RUP setting (INT-RUP) as it is based on sponge structure [17]. However, it does not provide privacy when used in RUP settings. We have considered PA1 and PA2 security notion from [35] along with IND-CPA to provide privacy in RUP scenario.

We have chosen SAEB for our work as it is very recent lightweight nonce based AE scheme which requires minimum state size along with other functionalities. In table10, we have showed a comparison of various lightweight block cipher based AE schemes

Our contribution: In this work, We have shown a simple PA attack on SAEB. Then we have proposed a modification to SAEB to overcome this attack. We have proposed two modified versions of SAEB called RSAEB : first one provides PA1 security in nonce respecting scenario and another one provides PA1 in nonce misuse scenario and PA2. PA2

	Ref	Online	Inverse-free	State size	Rate	Static AD
JAMBU	[47]	✓	✓	$1.5n$	$1/2$	✗
SILC	[75]	✓	✓	$2n$	$1/2$	✗
CLOC	[74]	✓	✓	$2n$	$1/2$	✓
COFB	[23]	✓	✓	$1.5n$	1	✗
OTR	[59]	✓	✓	$4n$	1	✓
SUNDAE	[6]	✗	✓	n	$1/2$	✓
SAEB	[60]	✓	✓	n	$1/2$	✓

Table 10.1: Comparison of various Block-cipher based AE's considering n is the block cipher size.

is stronger security notion than PA1, which comes at the cost of an additional pass.

10.1 Authenticated Encryption

We consider deterministic authenticated encryption in the context of potential release of unverified plaintext (RUP). Following Andreeva et al. [35], we separate the decryption algorithm into plaintext computation and tag verification. Formally, let $(k, t) \in \mathbb{N}$. An authenticated encryption scheme AE is a family of algorithms $(\mathcal{E}, \mathcal{D}, \mathcal{V})$ where

$$\begin{aligned} \mathcal{E}_K &: (N, A, M) \to (C, T) \\ \mathcal{D}_K &: (N, A, C, T) \to M \\ \mathcal{V}_K &: (N, A, C, T) \to \top/\bot \end{aligned}$$

Here, $K \in \{0,1\}^k$ is a key, $A \in \{0,1\}^*$ associated data, $M \in \{0,1\}^*$ a message, $T \in \{0,1\}^t$ the verification tag, and $C \in \{0,1\}^{|M|}$ the ciphertext. The authenticated encryption scheme is required to be correct, i.e., should satisfied that $\mathcal{D}_K(N, A, \mathcal{E}_K(N, A, M)) = M$ and $\mathcal{V}_K(N, A, \mathcal{E}_K(N, A, M)) = \top$ for any K, N, A, M.

10.1.1 Conventional Security Models

Conventionally, an authenticated encryption scheme should offer confidentiality, meaning that its ciphertexts are computationally indistinguishable from random, and integrity, meaning that its tags are unforgeable. For confidentiality, we stay with the standard security model that measures the adversarial power to distinguish $\mathcal{E}_K$ for random key $K \xleftarrow{\$} \{0,1\}^k$ from a random function $\$ \xleftarrow{\$} F(,t)$.

Definition 16. *Let $AE = (\mathcal{E}, \mathcal{D}, \mathcal{V})$ be an authenticated encryption scheme. The confidentiality or privacy of* AE *against an adversary $\mathcal{A}$ is defined as*

$$\mathrm{Priv}_{AE}(\mathcal{A}) = Pr(\mathcal{A}^{\mathcal{E}_K} \to 1) - Pr(\mathcal{A}^{\$} \to 1)$$

where randomness is taken over $K \xleftarrow{\$} \{0,1\}^k$, $\$ \xleftarrow{\$} F(,t)$ and over random choices of $\mathcal{A}$.*

For integrity, the adversary is given access to $\mathcal{E}_K$ for a random key $K \xleftarrow{\$} \{0,1\}^k$ and it additionally gets access to verification oracle $\mathcal{V}_K$, and it succeeds if it manages to find a forgery, i.e., oracle returns $\top$ on input (A, C, T) that was not the result of an earlier encryption query.

Definition 17. *Let $AE = (\mathcal{E}, \mathcal{D}, \mathcal{V})$ be an authenticated encryption scheme. The integrity of* AE *against an adversary $\mathcal{A}$ is defined as*

$$\mathrm{INT}_{AE}(\mathcal{A}) = Pr(\mathcal{A}^{\mathcal{E}_K, \mathcal{V}_K} forges)$$

where randomness is taken over $K \xleftarrow{\$} \{0,1\}^k$ and over the random choices of $\mathcal{A}$. The event "forges" happens if $\mathcal{V}_K$ returns $\top$ for an input (A, C, T) and (A, C, T) is not the result of an earlier encryption query.

Typically, we combine the above two security notions of an authenticated encryption into one, which is defined as follows:

Definition 18. *Let $AE = (\mathcal{E}, \mathcal{D}, \mathcal{V})$ be an authenticated encryption scheme. The* AE *security of AE against an adversary $\mathcal{A}$ is defined as*

$$\mathrm{AE}_{AE}(\mathcal{A}) = Pr(\mathcal{A}^{\mathcal{E}_K, \mathcal{V}_K} \to 1) - Pr(\mathcal{A}^{\$, \perp} \to 1)$$

where randomness is taken over $K \xleftarrow{\$} \{0,1\}^k$, $\$ \xleftarrow{\$} F(,t)$ and over random choices of $\mathcal{A}$, and $\perp$ is an oracle that rejects on every input.*

10.1.2 RUP Security Model

Andreeva et al. [35] proposed two versions for confidentiality in the RUP setting: plaintext awareness 1 (PA1) and plaintext awareness 2 (PA2). In both models, the real world is constituted of both $\mathcal{E}_K$ and $\mathcal{D}_K$ for random $K \leftarrow \{0,1\}^k$. In the ideal world, the decryption

algorithm is replaced by a simulator $\mathcal{S}$ that mimics the outputs of $\mathcal{D}_K$. In PA1, the simulator has insight in the queries that the adversary made to $\mathcal{E}_K$, whereas in PA2, it has no insight in these queries. We will consider PA1 security, where the simulator has insight to the queries that the adversary makes to $\mathcal{E}_K$.

Definition 19. *Let $AE = (\mathcal{E}, \mathcal{D}, \mathcal{V})$ be an authenticated encryption scheme. Let $\mathcal{S}$ be a simulator. The* PA1 *security of* AE *against an adversary $\mathcal{A}$ is defined as*

$$\text{PA1}^{\mathcal{S}}_{AE}(\mathcal{A}) = Pr(\mathcal{A}^{\mathcal{E}_K, \mathcal{D}_K} \to 1) - Pr(\mathcal{A}^{\mathcal{E}_K, \mathcal{S}} \to 1)$$

where randomness is taken over $K \xleftarrow{\$} \{0,1\}^k$ and over random choices of $\mathcal{A}$.

We consider integrity in the RUP setting, which differs from the conventional notion of integrity in the fact that the adversary has access to the decryption algorithm, and can use its outputs to improve its advantage in forging a tag. This notion is taken from Andreeva et al. [35].

Definition 20. *Let $AE = (\mathcal{E}, \mathcal{D}, \mathcal{V})$ be an authenticated encryption scheme. The* $\text{INT} - \text{RUP}$ *of* AE *against an adversary $\mathcal{A}$ is defined as*

$$\text{INT} - \text{RUP}_{AE}(\mathcal{A}) = Pr(\mathcal{A}^{\mathcal{E}_K, \mathcal{D}_K, \mathcal{V}_K} forges)$$

where randomness is taken over $K \xleftarrow{\$} \{0,1\}^k$ and over the random choices of $\mathcal{A}$. The event "forges" happens if $\mathcal{V}_K$ returns $\top$ for an input (A, C, T) which is not the result of an earlier encryption query.

10.2 Generalized AE Security in RUP Settings

In this section, we define an indistinguishability framework for AE security in the released unverified plaintext setting, dubbed as AERUP in [24]. The new model, in itself, basically combines RUP confidentiality (i.e., PA1) and integrity (i.e., INT-RUP), in the exact same way as AE combined Priv and INT (see Definition 18). We have directly taken this result from [24]. As before, the security model considers a distinguisher that has access to either of two worlds, the real or ideal world. In the real world, it can query the encryption, decryption, and verification oracles of the AE algorithm, $(\mathcal{E}, \mathcal{D}, \mathcal{V})$ for random key $K \xleftarrow{\$} \{0,1\}^k$. In the ideal world, it has access to a random function $\$ \xleftarrow{\$} F(*, *, *)$, a simulator $\mathcal{S}$ that has access to the history of encryption queries, and a reject oracle $\perp$.

Definition 21. *Let $AE = (\mathcal{E}, \mathcal{D}, \mathcal{V})$ be a nonce based authenticated encryption scheme. Let $\mathcal{S}$ be a simulator. The AERUP security of* AE *against a nonce-respecting adversary $\mathcal{A}$ is defined as*

$$\text{AERUP}^{\mathcal{S}}_{AE}(\mathcal{A}) = Pr(\mathcal{A}^{\mathcal{E}_K, \mathcal{D}_K, \mathcal{V}_K} \to 1) - Pr(\mathcal{A}^{\$, \mathcal{S}, \perp} \to 1)$$

where randomness is taken over $K \xleftarrow{\$} \{0,1\}^k$, $\$ \xleftarrow{\$} F(*,*,*)$ *and over random choices of* $\mathcal{S}$ *and* $\mathcal{A}$. *The adversary is not allowed to query a response from the first oracle to the third oracle. Note that, as the simulator has access to the query history that an adversary* $\mathcal{A}$ *made to its first oracle, we can safely allow* $\mathcal{S}$ *to query an earlier response from the first oracle to the second oracle [24].*

10.2.1 Reductions from AERUP

The model of AERUP security is general: if an authenticated encryption scheme AE is AERUP secure, then it is AE secure in terms of Definition 18, PA1 secure in terms of Definition 19, and INT-RUP secure in terms of Definition 20. Stated differently.

$$\mathrm{AE} + \mathrm{PA1} + \mathrm{INT} - \mathrm{RUP} \Leftarrow \mathrm{AERUP}$$

We have directly taken this result from [24]. All the reductions proof to support the following propositions can be directly taken from [24].

Proposition 5. *[24] (*$\mathrm{AE} \Leftarrow \mathrm{AERUP}$*). Let* $AE = (\mathcal{E}, \mathcal{D}, \mathcal{V})$ *be an authenticated encryption scheme. Let* $\mathcal{S}$ *be a simulator. For any adversary* $\mathcal{A}$ *with query complexity* q *which is the sum of encryption and verification query complexities,*

$$\mathrm{AE}_{AE}(\mathcal{A}) \leq \mathrm{AERUP}^{\mathcal{S}}_{AE}(\mathcal{B}),$$

where $\mathcal{B}$ *is some adversary with query complexity* q.

Proposition 6. *[24] (*$\mathrm{PA1} \Leftarrow \mathrm{AERUP}$*). Let* $AE = (\mathcal{E}, \mathcal{D}, \mathcal{V})$ *be an authenticated encryption scheme. Let* $\mathcal{S}$ *be a simulator. For any adversary* $\mathcal{A}$ *that makes* q_e *encryption queries and* q_d *decryption queries,*

$$\mathrm{PA1}^{\mathcal{S}}_{AE}(\mathcal{A}) \leq \mathrm{AERUP}^{\mathcal{S}}_{AE}(\mathcal{B}_1) + \mathrm{Priv}_{AE}(\mathcal{B}_2),$$

where $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *are some adversaries where the query complexity of* $\mathcal{B}_1$ *is* $q_e + q_d$ *and the query complexity of* $\mathcal{B}_2$ *is* q_e.

Proposition 7. *[24] (*$\mathrm{INT} - \mathrm{RUP} \Leftarrow \mathrm{AERUP}$*). Let* $AE = (\mathcal{E}, \mathcal{D}, \mathcal{V})$ *be an authenticated encryption scheme. Let* $\mathcal{S}$ *be a simulator. For any adversary* $\mathcal{A}$ *that makes altogether* q *encryption, decryption, and verification queries,*

$$\mathrm{INT} - \mathrm{RUP}_{AE}(\mathcal{A}) = \mathrm{AERUP}^{\mathcal{S}}_{AE}(\mathcal{B}),$$

where $\mathcal{B}$ *is some adversary with query complexity* q *that involves encryption, decryption and verification queries, and* $\mathcal{S}$ *is any simulator.*

10.3 Description of SAEB

For positive integers n and k, let $E \in BC(\{0,1\}^k, \{0,1\}^n)$ be the underlying block-cipher. Positive integers r_1, r_2, r are the sizes of an associated data block, a nonce and a plaintext block respectively such that $1 \leq r_1 \leq n-2$, $1 \leq r_2 \leq n-2$ and $1 \leq r \leq n-2$. A positive integer τ is the tag size. Let $c_1 = n - r_1$, $c_2 = n - r_2$ and $c = n - r$. The sets $\mathcal{K}, \mathcal{N}, \mathcal{M}, \mathcal{C}, \mathcal{A}$ and $\mathcal{T}$ are defined as $\mathcal{K} = \{0,1\}^k$, $\mathcal{N} = \{0,1\}^{r_2}$, $\mathcal{M} = \{0,1\}^* \setminus \{\lambda\}$, $\mathcal{C} = \{0,1\}^* \setminus \{\lambda\}$, $\mathcal{A} = \{0,1\}^*$ and $\mathcal{T} = \{0,1\}^\tau$. SAEB uses a one-zeros padding: for a positive integer r^* and a bit string x with $0 < |x| < r^*$, $x||10^*r^*$ denotes a bit string where a bit 1 is appended to x, and then a minimum number of bits 0 is appended so that the length in bits becomes r^*.

SAEB is a nonce based authenticated encryption scheme, which takes associated data $A \in \mathcal{A}$, nonce $N \in \mathcal{N}$ and plaintext $M \in \mathcal{M}$ as input, and produces ciphertext tag pair (C, T) as output. Decryption takes N, A, C, T as input and returns M or $\perp$ depending upon the verification. Schematic representation of SAEB is shown in fig. 10.1.

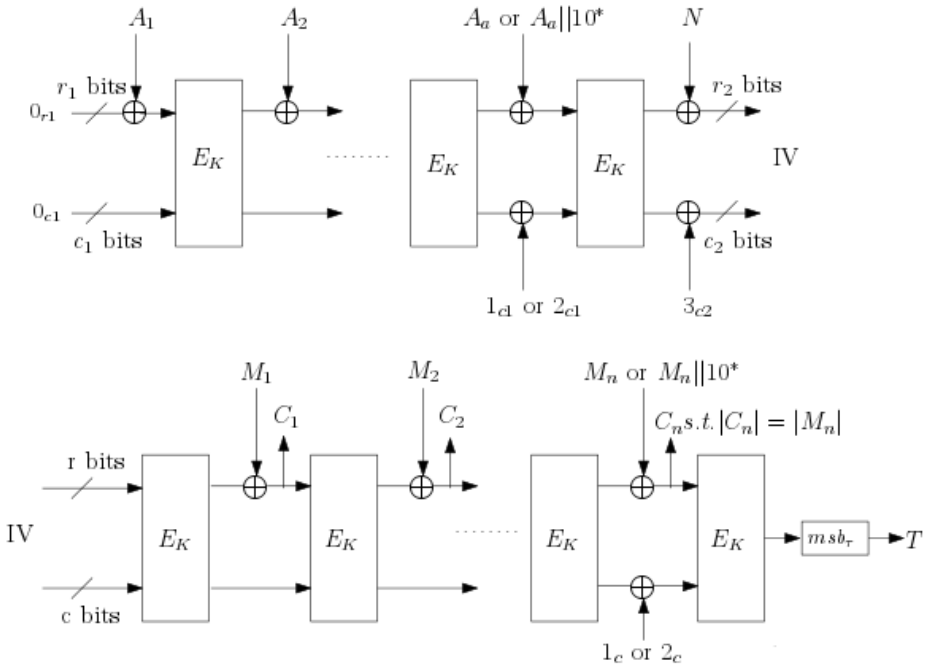


Figure 10.1: SAEB Encryption [60]

10.4 PA1 Attack on SAEB

In this section, we have proposed a simple PA1 attack on SAEB . In this settings, decryption algorithm is allowed to release a unverified plaintext prior to tag verification. As given in definition 5, the goal of PA1 adversary is to distinguish between the actual decryption oracle and extractor which has access to history of encryption queries. In PA1 settings, adversary

D has access to the actual encryption oracle $\mathcal{E}nc_K$ and oracle $\mathcal{O}_2$ which can either be actual decryption algorithm or an extractor.

Following are the steps for the attack:

1. Adversary D generates C, T uniformly at random from $\{0,1\}^n$ and makes a query to $\mathcal{O}_2(N, A, C, T)$ where N and A are chosen randomly , and gets M as an output irrespective of the tag verification.

2. Then D makes a query to encryption oracle $\mathcal{E}nc_K(N, A, M)$ and receives C', T.

3. The adversary D outputs 1 if C' is exactly same as C, otherwise outputs 0, irrespective of value of T.

10.5 RSAEB v1

In this section, we have proposed a modification to SAEB scheme. Our modified scheme is shown in fig. 10.2. It has following advantages over existing SAEB:
- Provides privacy in RUP setting with nonce respecting scenario.
- Associated data can be processed in parallel along with plain text.

Our authenticated encryption scheme takes secret nonce N, associated data A and a plaintext M as input and produces (FV, C, T) as output. Decryption algorithm takes (A, FV, C, T) as input and produces M or $\perp$ depending upon the verification. If a tag verifies then it returns M, otherwise $\perp$. Detailed encryption and decryption procedure for this proposed scheme is given in algorithm 7 and 29 respectively.

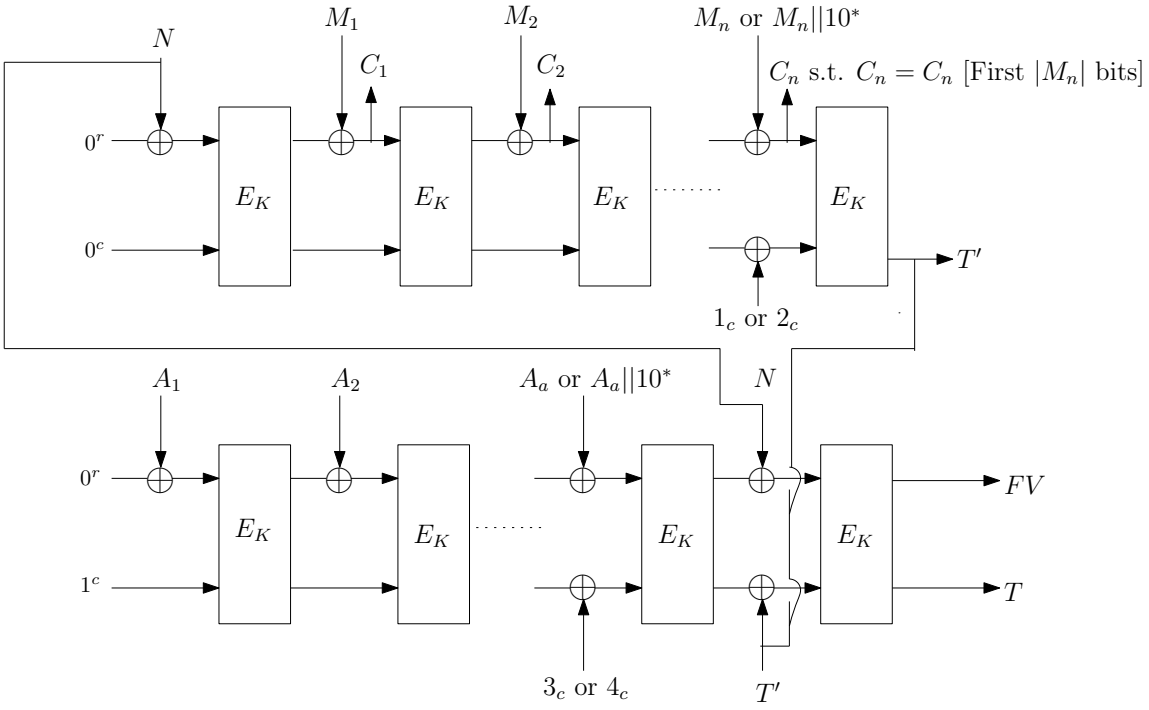


Figure 10.2: RSAEB v1

The choices of constants in this design are highly related to the goal of security. The initialization constants in processing of associated data ($0^r||1^c$) and for message ($0^r||0^c$) has chosen in a way that collision event would happen only with negligible probability. Different choices of constant for processing the last block of associated data and message has been taken to avoid collision on any internal state values. In the processing of message, if the last block is of length r, then the constant value 1_c is used, otherwise constant value 2_c is used. For the processing of associated data, if last block is of length r, then constant value 3_c is used, otherwise constant value 4_c is used.

Algorithm 7: RSAEB v1 Encryption $\mathcal{E}_K(N, A, M)$

$S \leftarrow (0^r \oplus N)||0^c$
$S \leftarrow E_K(S)$
if $M \neq \phi$ **then**
  $M_1||M_2 \ldots M_n \xleftarrow{r} M$
  for $i = 1$ *to* $n-1$ **do**
    $C_i \leftarrow S_r \oplus M_i$
    $S \leftarrow E_K(C_i||S_c)$
  if $|M_n| = r$ **then**
    $S \leftarrow S \oplus (M_n||1_c)$
    $C_n \leftarrow S_r$
  else
    $S \leftarrow S \oplus ((M_n||10^*)||2_c)$
    $C_n \leftarrow (S)^{|M_n|}$
  $S \leftarrow E_K(S)$
  $C \leftarrow C_1||C_2 \ldots||C_n$
else
  $S \leftarrow E_K((S_r \oplus 10^{r-1})||(S_c \oplus 2_c))$
  $C \leftarrow \phi$
$T' \leftarrow S_c$
$S \leftarrow 0^r||1^c$
if $A \neq \phi$ **then**
  $A_1||A_2 \ldots A_a \xleftarrow{r} A$
  for $i = 1$ *to* $a-1$ **do**
    $S \leftarrow E_K(S_r \oplus A_i||S_c)$
  if $|A_a| == r$ **then**
    $S \leftarrow E_K((S_r \oplus A_a)||(S_c \oplus 3_c))$
  else
    $S \leftarrow E_K((S_r \oplus A_a||10^*)||(S_c \oplus 4_c))$
else
  $S \leftarrow E_K((S_r \oplus 10^{r-1})||(S_c \oplus 4_c))$
$S \leftarrow E_k((S_r \oplus N)||(S_c \oplus T'))$
$FV \leftarrow S_r, T \leftarrow S_c$
Return (C, FV, T)

Algorithm 8: RSAEB v1 Decryption $\mathcal{D}_K(A, C, FV, T)$

$S \leftarrow 0^r||1^c$
if $A \neq \phi$ **then**
  $A_1||A_2 \ldots A_a \xleftarrow{r} A$
  for $i = 1$ *to* $a-1$ **do**
    $S \leftarrow E_K(S_r \oplus A_i||S_c)$
  if $|A_a| == r$ **then**
    $S \leftarrow E_K((S_r \oplus A_a)||(S_c \oplus 3_c))$
  else
    $S \leftarrow E_K((S_r \oplus A_a||10^*)||(S_c \oplus 4_c))$
else
  $S \leftarrow E_K((S_r \oplus 10^{r-1})||(S_c \oplus 4_c))$
$S' \leftarrow E_K^{-1}(FV||T)$
$N \leftarrow S_r \oplus S'_r$
$T' \leftarrow S_c \oplus S'_c$
$S \leftarrow (0^r \oplus N)||0^c$
$C_1||C_2 \ldots C_n \xleftarrow{r} C$
for $i = 1$ *to* $n-1$ **do**
  $M_i \leftarrow S_r \oplus C_i$
  $S \leftarrow E_K(C_i||S_c)$
if $|C_n| = r$ **then**
  $M_n = C_n \oplus S_r$
  $S \leftarrow S \oplus (M_n||1_c)$
else
  $M_n \leftarrow (S)^{|C_n|} \oplus C_n$
  $S \leftarrow S \oplus ((M_n||10^*)||2_c)$
$S \leftarrow E_K(S)$
$M \leftarrow M_1||M_2 \ldots||M_n$
$T'' \leftarrow S_c$
if $(T'' == T')$ **then**
  Return M
else
  Return $\perp$

10.6 RSAEB v2

In this section, we proposed a another version of the modified scheme. The idea behind this version is to make it PA2 secure and PA1 secure in nonce misuse settings. PA2 is strengthening of PA1 where extractor has no longer access to the query history of $\mathcal{E}_K$. This comes at the cost of an extra pass. Our idea is t use the ciphertext to hide the nonce in such a way that recovering it properly requires that no change was made to the ciphertext. As a result, if a change did occur, it would affect the nonce, which, used by decryption algorithm, would decrypt the ciphertext into a meaningless data. The proposed construction is shown in Fig. 10.3 and the detailed encryption and decryption procedure is given in algorithm 9 and 10 respectively.

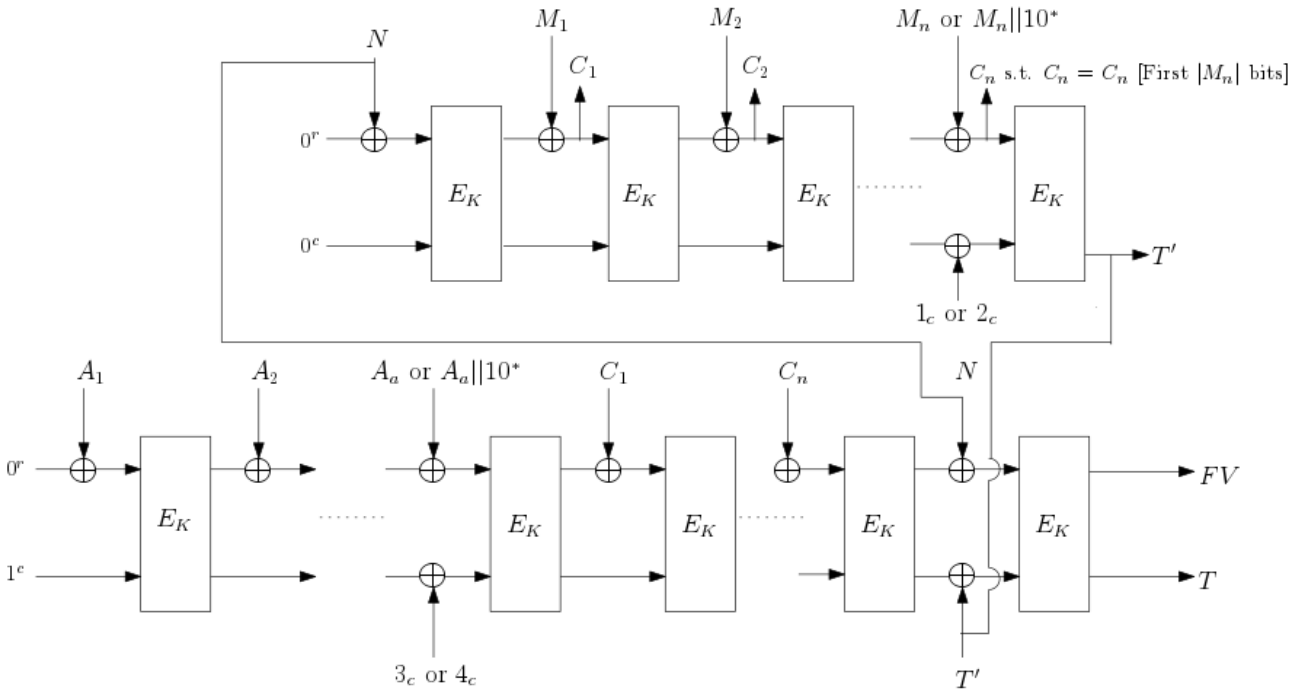


Figure 10.3: RSAEB v2

Algorithm 9: RSAEB v2 Encryption $\mathcal{E}_K(N, A, M)$

$S \leftarrow (0^r \oplus N)||0^c$
$S \leftarrow E_K(S)$
if $M \neq \phi$ **then**
  $M_1||M_2 \ldots M_n \xleftarrow{r} M$
  for $i = 1$ *to* $n-1$ **do**
    $C_i \leftarrow S_r \oplus M_i$
    $S \leftarrow E_K(C_i||S_c)$
  if $|M_n| = r$ **then**
    $S \leftarrow S \oplus (M_n||1_c)$
    $C_n \leftarrow S_r$
  else
    $S \leftarrow S \oplus ((M_n||10^*)||2_c)$
    $C_n \leftarrow (S)^{|M_n|}$
  $S \leftarrow E_K(S)$
  $C \leftarrow C_1||C_2 \ldots||C_n$
else
  $S \leftarrow E_K((S_r \oplus 10^{r-1})||(S_c \oplus 2_c))$
  $C \leftarrow \phi$
$T' \leftarrow S_c$
$S \leftarrow 0^r||1^c$
if $A \neq \phi$ **then**
  $A_1||A_2 \ldots A_a \xleftarrow{r} A$
  for $i = 1$ *to* $a-1$ **do**
    $S \leftarrow E_K(S_r \oplus A_i||S_c)$
  if $|A_a| == r$ **then**
    $S \leftarrow E_K((S_r \oplus A_a)||(S_c \oplus 3_c))$
  else
    $S \leftarrow E_K((S_r \oplus A_a||10^*)||(S_c \oplus 4_c))$
else
  $S \leftarrow E_K((S_r \oplus 10^{r-1})||(S_c \oplus 4_c))$
for $i = 1$ *to* n **do**
  $S \leftarrow E_K(S_r \oplus C_i||S_c)$
$S \leftarrow E_k((S_r \oplus N)||(S_c \oplus T'))$
$FV \leftarrow S_r, T \leftarrow S_c$
Return (C, FV, T)

Algorithm 10: RSAEB v2 Decryption $\mathcal{D}_K(A, C, FV, T)$

$S \leftarrow 0^r||1^c$
if $A \neq \phi$ **then**
  $A_1||A_2 \ldots A_a \xleftarrow{r} A$
  for $i = 1$ *to* $a-1$ **do**
    $S \leftarrow E_K(S_r \oplus A_i||S_c)$
  if $|A_a| == r$ **then**
    $S \leftarrow E_K((S_r \oplus A_a)||(S_c \oplus 3_c))$
  else
    $S \leftarrow E_K((S_r \oplus A_a||10^*)||(S_c \oplus 4_c))$
else
  $S \leftarrow E_K((S_r \oplus 10^{r-1})||(S_c \oplus 4_c))$
$C_1||C_2 \ldots C_n \xleftarrow{r} C$
for $i = 1$ *to* n **do**
  $S \leftarrow E_K(S_r \oplus C_i||S_c)$
$S' \leftarrow E_K^{-1}(FV||T)$
$N \leftarrow S_r \oplus S'_r$
$T' \leftarrow S_c \oplus S'_c$
$S \leftarrow (0^r \oplus N)||0^c$
$C_1||C_2 \ldots C_n \xleftarrow{r} C$
for $i = 1$ *to* $n-1$ **do**
  $M_i \leftarrow S_r \oplus C_i$
  $S \leftarrow E_K(C_i||S_c)$
if $|C_n| = r$ **then**
  $M_n = C_n \oplus S_r$
  $S \leftarrow S \oplus (M_n||1_c)$
else
  $M_n \leftarrow (S)^{|C_n|} \oplus C_n$
  $S \leftarrow S \oplus ((M_n||10^*)||2_c)$
$S \leftarrow E_K(S)$
$M \leftarrow M_1||M_2 \ldots||M_n$
$T'' \leftarrow S_c$
if $(T'' == T')$ **then**
  Return M
else
  Return $\perp$

10.7 Security of RSAEB v1

We have provided the proof of RSAEB v1 construction in AERUP security model proposed by chang et. al in [24]. We have chosen AERUP model by chang et. al in [24] as it combines the PA1 and INT-RUP security notion. The model of AERUP security is general: if an authenticated encryption scheme AE is AERUP secure, then it is AE secure in terms of Definition 18, PA1 secure in terms of Definition 19, and INT-RUP secure in terms of Definition 20. There is an another security model called RUPAE proposed by Barewell et.al in [7] to provide security in RUP settings, however it combines the PA2 and INT-RUP security notions. As we have chosen PA1 security of RSAEB v1, we will use the AERUP model by chang et. al in [24] to provide the proof. Considering a information-theoretic model, we have replaced a underlying keyed blockcipher E_K where $K \leftarrow \{0,1\}^k$ with a random permutation P where $P \leftarrow Perm(\{0,1\}^n)$. To provide a proof, we have considered a nonce-respecting adversary i.e. he always makes the query with unique nonce value.

The upper bound of the security advantage is given in following theorem. The security proof is given in next subsection.

Theorem 8. *(*AERUP *security of RSAEB v1). Let* $\Pi = (\mathcal{E}, \mathcal{D}.\mathcal{V})$ *be the RSAEB v1 authenticated encryption scheme based on block cipher* $E : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$. *Let* $\mathcal{A}$ *be a nonce-respecting adversary that makes* $q_{\mathcal{E}}$ *encryption queries,* $q_{\mathcal{D}}$ *decryption queries and* $q_{\mathcal{V}}$ *verification queries. Let* $\sigma_{\mathcal{E}}$, $\sigma_{\mathcal{D}}$ *and* $\sigma_{\mathcal{V}}$ *be total number of blockcipher calls with distinct input during encryption, decryption and verification queries. Let* σ *be the total number of blockcipher calls with distinct inputs by all queries. The the AERUP advantage of an adversary* $\mathcal{A}$ *can be given as:*

$$\mathrm{Adv}_{\Pi}^{\mathrm{AERUP}}(\mathcal{A}) \leq \binom{\sigma}{2}/2^b + \frac{q_{\mathcal{D}}}{2^r} + \frac{11\sigma^2}{2^c}.$$

10.7.1 Proof

We consider any adversary that has access to either real world or ideal world and tries to distinguish both worlds. The adversary has encryption complexity $\sigma_{\mathcal{E}}$, decryption complexity $\sigma_{\mathcal{D}}$, and verification complexity $\sigma_{\mathcal{V}}$, with $\sigma_{\mathcal{E}} + \sigma_{\mathcal{D}} + \sigma_{\mathcal{V}} = \sigma$, and operates in time t. For the simplicity of proof, we do not allow an adversary to make repeated queries which means that all queries are distinct.

Description of Real World

The real world consists of the encryption oracle $\Pi.\mathcal{E}[P]$, the decryption oracle $\Pi.\mathcal{D}[P]$ and the verification oracle $\Pi.\mathcal{V}[P]$ where $P \leftarrow Perm(\{0,1\}^n)$. A set I_B is maintained which stores the tuple $(N_i, A_i(1 \ldots a_i), M_i(1 \ldots n_i), C_i(1 \ldots n_i), FV_i, T_i)$ for each encryption and decryption query. Along with this, another set I_P is maintained which stores the tuple (U_j, V_j) corresponding to each internal query to the permutation P.

Description of Ideal World

The ideal world consists of three oracles $(\$, \mathcal{S}, \perp)$. The verification oracle $\perp$ simply responds with the $\perp$ sign for each input (A, C, FV, T). We will elaborate on the remaining two oracles, encryption $\$$ and decryption $\mathcal{S}$, in detail. This is just to note that, as we work in the PA1 setting, i.e., $\mathcal{S}$ will have access to the queries previously made to the encryption oracle. The encryption oracle $\$$ is a random function that for each input $(N_i, A_i, M_i) = (N_i, A_i[1 \dots a_i], M_i[1 \dots n_i])$, it generates a (C_i, FV_i, T_i) as following:

$$\begin{aligned} C_i = C_i[1 \dots n_i] &\xleftarrow{\$} \{0,1\}^{|M_i|} \\ FV_i &\xleftarrow{\$} \{0,1\}^n \\ T_i &\xleftarrow{\$} \{0,1\}^n \end{aligned}$$

I_B set is updated by adding query response as a tuple $(N_i, A_i(1 \dots a_i), M_i(1 \dots n_i), C_i(1 \dots n_i), FV_i, T_i)$, after each query.

The decryption oracle $\mathcal{S}$ is a simulator that we define to operate as shown in algorithm 11 on input of a query (A^*, C^*, FV^*, T^*). Simulator has a access to the set I_B maintained earlier. Lets denote $A^* = (A^*_1 \dots A^*_{a^*})$, $C^* = (C^*_1 \dots C^*_{n^*})$ and the number of blocks of C^* by n^*.

Algorithm 11: Simulator $\mathcal{S}$ on input $(A^*(1 \dots a^*), C^*(1 \dots n^*), FV^*, T^*)$

Set $U^f \leftarrow 0^r || 0^c, V^f = (V^f_r || V^f_c) \leftarrow 0^r || 1^c, d \leftarrow 0^c$
if $\exists N, M$ *s.t.* $(N, A^*, M, C^*, FV^*, T^*) \in I_B$ **then**
  set $M^* \leftarrow M$, Return M^*, EXIT
if $\exists (N, M(1 \dots n), C(1 \dots n))$ *s.t.* $(N, A^*, M_1 \dots M_n, C^*_1 \dots C^*_j C_{j+1} \dots C_{n^*}, FV^*, T^*) \in I_B$ *for* $0 < j \leq n^*$ *and* $C^*_i = C_i$ *for* $i = 1$ *to* j **then**
  if $j = n^*$ **then**
    $M^*_i = M_i$ for $i = 1 \dots j$
  else
    if $j = n^* - 1$ **then**
      $M^*_i = M_i$ for $i = 1 \dots j$, $M^*_{n^*} = C^*_{n^*} \oplus |C_{j+1}|^{|C^*_{n^*}|} \oplus |M_{j+1}|^{|C^*_{n^*}|}$
    else
      $M^*_i = M_i$ for $i = 1 \dots j$,
      $M^*_{j+1} = C^*_{j+1} \oplus C_{j+1} \oplus M_{j+1}$,
      $M^*_{j+2} \dots M^*_{n^*-1} \xleftarrow{\$} \{0,1\}^r$, $M^*_{n^*} \xleftarrow{\$} \{0,1\}^{|C^*_{n^*}|}$
  $I_B = I_B \cup (N, A^*, M^*, C^*, FV^*, T^*)$
  Return $M^* = M^*_1 \dots M^*_{n^*}$, EXIT
$M^* \xleftarrow{\$} \{0,1\}^{|C^*|}$
Return M^*

Lets denote $\Pi := RSAEB$. In this subsection, we will upper-bound the following advantage function.

$$\mathrm{Adv}_{\Pi}^{\mathrm{AERUP}}(\mathcal{A}) = Pr[G_0] - Pr[G_8]$$

where Game G_0 represents the real world $G_0 := \big(P \leftarrow Perm(\{0,1\}^n); \mathcal{A}^{\Pi.\mathcal{E}[P],\Pi.\mathcal{D}[P],\Pi.\mathcal{V}[P]} \rightarrow 1\big)$ and G_8 represents the ideal world, $G_8 := (\mathcal{A}^{\$,\mathcal{S},\perp} \rightarrow 1)$.

The advantage of an adversary is his ability to distinguish the real world from an ideal world. We used the game based framework to compute this advantage. We define a sequence of eight games from G_0 to G_8, where Game G_0 represents the real world and G_8 represents ideal world. Hence, the advantage of an adversary can be given by:

$$\begin{aligned}\mathrm{Adv}_{\Pi}^{\mathrm{AERUP}}(\mathcal{A}) \;=\; & (Pr[G_0] - Pr[G_1]) \\ & +(Pr[G_1] - Pr[G_2]) + (Pr[G_2] - Pr[G_3]) \\ & +Pr[G_3] - Pr[G_4] + (Pr[G_4] - Pr[G_5]) \\ & +(Pr[G_5] - Pr[G_6]) + (Pr[G_6] - Pr[G_7]) \\ & +(Pr[G_7] - Pr[G_8])\end{aligned}$$

Computations for probability difference between consecutive games are provided below.

Game G_0: Game G_0 (shown in Fig. 10.4) perfectly simulates the real world.

$$Pr[P \leftarrow Perm(\{0,1\}^n); \mathcal{A}^{\Pi.\mathcal{E}[P],\Pi.\mathcal{D}[P],\Pi.\mathcal{V}[P]} \rightarrow 1] = Pr[A^{G0} \rightarrow 1]$$

Game G_1: Game G_1 (shown in Fig. 10.5 with boxed statements) is exactly same as game $G0$.

$$Pr[A^{G_0} \rightarrow 1] = Pr[A^{G_1} \rightarrow 1]$$

Game G_2: Game G_1 and G_2 (shown in Fig. 10.5 without boxed statements) differs only when bad event occurs, otherwise, in absence of bad event, both games are same.

$$\mid Pr[A^{G_1} \rightarrow 1] \text{-} Pr[A^{G_2} \rightarrow 1] \mid \leq Pr[bad]$$

The probability of having this bad event can be given by the bound of PRP-PRF switching lemma. Hence, we have

$$Pr[bad] \;\leq\; \binom{\sigma}{2}/2^n$$

Therefore,

$$|Pr[A^{G_1} \rightarrow 1] - Pr[A^{G_2} \rightarrow 1]| \;\leq\; \binom{\sigma}{2}/2^b$$

Game G_3: In game G_3 (shown in Fig. 10.6), decryption oracle is replaced by simulator $\mathcal{S}$, which tries to mimic the actual decryption oracle. Simulator $\mathcal{S}$ have an access to the history of previous encryption queries i.e. set I_B made by an adversary .
In this case, if prefix of decryption query matches with any encryption query present in history then simulator can trivially answer the output corresponding to the common prefix and can compute one more block of plaintext. After that, rest of the plaintext block will be generated randomly. In case, no common prefix is present, simulator will output the random plaintext of same length as ciphertext. The difference between G_2 and G_3 lies in presence of bad events. These bad events can be of two types.

- bad_1: This event can be defined as a nonce repeating event. Lets assume for a fixed N^*, C^* in set I_B, adversary choose (A, FV, T) in such a way that the newly generated nonce N collides with existing nonce N^* in set I_B. Hence, the probability of having this bad_1 event can be given by the probability of collision on r bits. Therefore, we have

$$Pr[bad_1] \leq \frac{q_{\mathcal{D}}}{2^r}$$

  where $q_{\mathcal{D}}$ is total number of decryption queries.

- bad_2: This bad event can be defined as a collision event on c bits. This can happen in a case when c bits (after prefix match and one more block computation) collides with the c bits of another query where rest of the C^* is same. Hence,

$$Pr[bad_1] \leq \frac{\sigma^2}{2^c}$$

Hence,

$$|Pr[A^{G_2} \rightarrow 1] - Pr[A^{G_3} \rightarrow 1]| \leq Pr[bad_1] + Pr[bad_2]$$
$$|Pr[A^{G_2} \rightarrow 1] - Pr[A^{G_3} \rightarrow 1]| \leq \frac{q_{\mathcal{D}}}{2^r} + \frac{\sigma^2}{2^c}$$

Game G_4: Game G_4 (shown in Fig. 10.7) is exactly same as G_3. It just expands the definition of verification oracle.

$$|Pr[A^{G_3} \rightarrow 1] - Pr[A^{G_4} \rightarrow 1]| \leq 0$$

Game G_5: Game G_4 and G_5 (shown in Fig. 10.8) differs only when bad events occurs, otherwise, in absence of bad event, both games are same.

$$\mid Pr[A^{G_4} \rightarrow 1]\text{-}Pr[A^{G_5} \rightarrow 1] \mid \leq Pr[bad_3] + Pr[bad_4]$$

- bad_3: This bad event happens when an adversary gets success in generating valid (A, C, FV, T) pair which has not been queried before or has not appeared as a output of previous encryption queries. Therefore, the probability of this bad event can be defined by computing the probability of getting collision on T' which is of c bits. Hence

$$Pr[bad_3] \leq \frac{\sigma^2}{2^c}$$

- bad_4: This bad event can de defined when an adversary gets a collision on c bits by reordering or recycling an existing permutation queries which is set I_P in a way such that c bits of the permutation output(V) from a existing pair (U, V) collides with the c bits of the permutation input (U') from some other existing pair (U', V') in set I_P. Hence,

$$Pr[bad_4] \leq \frac{9\sigma^2}{2^c}$$

Therefore,

$$\begin{aligned} |Pr[A^{G_4} \to 1] - Pr[A^{G_5} \to 1]| &\leq \frac{\sigma^2}{2^c} + \frac{9\sigma^2}{2^c}, \\ |Pr[A^{G_4} \to 1] - Pr[A^{G_5} \to 1]| &\leq \frac{10\sigma^2}{2^c} \end{aligned}$$

Game G_6: Game G_6 (shown in Fig. 10.9) is exactly same as G_5 from adversarial point of view. In G_6, instead of setting flag in verification, it directly returns $\perp$. Hence.

$$|Pr[A^{G_5} \to 1] - Pr[A^{G_6} \to 1]| \leq 0$$

Game G_7: Game G_7 (shown in Fig. 10.10 with boxed statements) is exactly same as G_6. It just expands the definition of an encryption oracle and some redundancy has been added at line 15 and 33. Hence,

$$|Pr[A^{G_7} \to 1] - Pr[A^{G_8} \to 1]| \leq 0$$

Game G_8: In Game G_8 (shown in Fig. 10.10 without boxed statements), instead of making P query, encryption oracle itself generates random output. Hence, It always returns random (C, FV, T).

$$|Pr[A^{G_7} \to 1] - Pr[A^{G_8} \to 1]| \leq 0$$

Therefore, by considering all the games, the upper bound of AERUP security of RSAEB can be given by:

$$\begin{aligned} \mathrm{Adv}_{\Pi}^{\mathrm{AERUP}}(\mathcal{A}) &\leq \binom{\sigma}{2}/2^b + \frac{q_{\mathcal{D}}}{2^r} + \frac{\sigma^2}{2^c} + \frac{10\sigma^2}{2^c} \\ &\leq \binom{\sigma}{2}/2^b + \frac{q_{\mathcal{D}}}{2^r} + \frac{11\sigma^2}{2^c} \end{aligned}$$

Initialize $I_B = \phi, I_P = \phi$, $S = S_c||S_r \leftarrow 0^r||0^c$

Encryption $\mathcal{E}_K(N, A, M)$

1 $S \leftarrow (0^r \oplus N)||0^c$
2 $S \leftarrow P(S)$
3 **if** $M \neq \phi$ **then**
4 $\quad M_1||M_2 \dots M_n \xleftarrow{r} M$
5 $\quad$ **for** $i = 1$ *to* $n-1$ **do**
6 $\quad\quad C_i \leftarrow S_r \oplus M_i$
7 $\quad\quad S \leftarrow P(C_i||S_c)$
8 $\quad$ **if** $|M_n| = r$ **then**
9 $\quad\quad S \leftarrow S \oplus (M_n||1_c)$
10 $\quad\quad C_n \leftarrow S_r$
11 $\quad$ **else**
12 $\quad\quad S \leftarrow S \oplus ((M_n||10^*)||2_c)$
13 $\quad\quad C_n \leftarrow (S)^{|M_n|}$
14 $\quad S \leftarrow P(S)$
15 $\quad C \leftarrow C_1||C_2 \dots ||C_n$
16 **else**
17 $\quad S \leftarrow P((S_r \oplus 10^{r-1})||(S_c \oplus 2_c))$
18 $\quad C \leftarrow \phi$
19 $T' \leftarrow S_c$
20 $S \leftarrow 0^r||1^c$
21 **if** $A \neq \phi$ **then**
22 $\quad A_1||A_2 \dots A_a \xleftarrow{r} A$
23 $\quad$ **for** $i = 1$ *to* $a-1$ **do**
24 $\quad\quad S \leftarrow P(S_r \oplus A_i||S_c)$
25 $\quad$ **if** $|A_a| == r$ **then**
26 $\quad\quad S \leftarrow P((S_r \oplus A_a)||(S_c \oplus 3_c))$
27 $\quad$ **else**
28 $\quad\quad S \leftarrow P((S_r \oplus A_a||10^*)||(S_c \oplus 4_c))$
29 **else**
30 $\quad S \leftarrow P((S_r \oplus 10^{r-1})||(S_c \oplus 4_c))$
31 $S \leftarrow P((S_r \oplus N)||(S_c \oplus T'))$
32 $FV \leftarrow S_r, T \leftarrow S_c$
33 $I_B = I_B \cup (N, A, M, C, FV, T)$
34 Return (C, FV, T)

On P **query on input** U

1 **if** $\exists V$ *s.t.* $(U, V) \in I_P$ **then**
2 $\quad$ return V
3 **else**
4 $\quad V \xleftarrow{\$} \{0,1\}^b$
5 $\quad$ **if** $\exists U'$ *s.t* $(U', V) \in I_P$ **then**
6 $\quad\quad V \xleftarrow{\$} \{0,1\}^b \setminus \{V : (*, V) \in I_P\}$
$\quad\quad I_P = I_P \bigcup \{(U, V)\}$
7 $\quad\quad$ return V;

Decryption $\mathcal{D}_K(A, C, FV, T)$

1 $S \leftarrow 0^r||1^c$
2 **if** $A \neq \phi$ **then**
3 $\quad A_1||A_2 \dots A_a \xleftarrow{r} A$
4 $\quad$ **for** $i = 1$ *to* $a-1$ **do**
5 $\quad\quad S \leftarrow P(S_r \oplus A_i||S_c)$
6 $\quad$ **if** $|A_a| == r$ **then**
7 $\quad\quad S \leftarrow P((S_r \oplus A_a)||(S_c \oplus 3_c))$
8 $\quad$ **else**
9 $\quad\quad S \leftarrow P((S_r \oplus A_a||10^*)||(S_c \oplus 4_c))$
10 **else**
11 $\quad S \leftarrow P((S_r \oplus 10^{r-1})||(S_c \oplus 4_c))$
12 $S' \leftarrow P^{-1}(FV||T)$
13 $N \leftarrow S_r \oplus S'_r$
14 $T' \leftarrow S_c \oplus S'_c$
15 $S \leftarrow (0^r \oplus N)||0^c$
16 $C_1||C_2 \dots C_n \xleftarrow{r} C$
17 **for** $i = 1$ *to* $n-1$ **do**
18 $\quad M_i \leftarrow S_r \oplus C_i$
19 $\quad S \leftarrow P(C_i||S_c)$
20 **if** $|C_n| = r$ **then**
21 $\quad M_n = C_n \oplus S_r$
22 $\quad S \leftarrow S \oplus (M_n||1_c)$
23 **else**
24 $\quad M_n \leftarrow (S)^{|C_n|} \oplus C_n$
25 $\quad S \leftarrow S \oplus ((M_n||10^*)||2_c)$
26 $S \leftarrow P(S)$
27 $M \leftarrow M_1||M_2 \dots ||M_n$
28 $I_B = I_B \cup (N, A, M, C, FV, T)$
29 Return M

Verification $\mathcal{V}_K(A, C, FV, T)$

1 Line 1 to 26 same as $\mathcal{D}_K(A, C, FV, T)$
2 $T'' \leftarrow S_c$
3 **if** $(T'' == T')$ **then**
4 $\quad$ Return $\top$
5 **else**
6 $\quad$ Return $\bot$

On P^{-1} **query on input** V

1 **if** $\exists U$ *s.t.* $(U, V) \in I_P$ **then**
2 $\quad$ return U
3 **else**
4 $\quad U \xleftarrow{\$} \{0,1\}^b$
5 $\quad$ **if** $\exists V'$ *s.t* $(U, V') \in I_P$ **then**
6 $\quad\quad U \xleftarrow{\$} \{0,1\}^b \setminus \{U : (U, *) \in I_P\}$
$\quad\quad I_P = I_P \bigcup \{(U, V)\}$
7 $\quad\quad$ return U;

Figure 10.4: G_0

Initialize $I_B = \phi, I_P = \phi$, $S = S_c||S_r \leftarrow 0^r||0^c$

Encryption $\mathcal{E}_K(N, A, M)$

$S \leftarrow (0^r \oplus N)||0^c$
$S \leftarrow P(S)$
if $M \neq \phi$ **then**
  $M_1||M_2 \dots M_n \xleftarrow{r} M$
  for $i = 1$ *to* $n-1$ **do**
    $C_i \leftarrow S_r \oplus M_i$
    $S \leftarrow P(C_i||S_c)$
  if $|M_n| = r$ **then**
    $S \leftarrow S \oplus (M_n||1_c)$
    $C_n \leftarrow S_r$
  else
    $S \leftarrow S \oplus ((M_n||10^*)||2_c)$
    $C_n \leftarrow (S)^{|M_n|}$
  $S \leftarrow P(S)$
  $C \leftarrow C_1||C_2 \dots ||C_n$
else
  $S \leftarrow P((S_r \oplus 10^{r-1})||(S_c \oplus 2_c))$
  $C \leftarrow \phi$
$T' \leftarrow S_c$
$S \leftarrow 0^r||1^c$
if $A \neq \phi$ **then**
  $A_1||A_2 \dots A_a \xleftarrow{r} A$
  for $i = 1$ *to* $a-1$ **do**
    $S \leftarrow P(S_r \oplus A_i||S_c)$
  if $|A_a| == r$ **then**
    $S \leftarrow P((S_r \oplus A_a)||(S_c \oplus 3_c))$
  else
    $S \leftarrow P((S_r \oplus A_a||10^*)||(S_c \oplus 4_c))$
else
  $S \leftarrow P((S_r \oplus 10^{r-1})||(S_c \oplus 4_c))$
$S \leftarrow P((S_r \oplus N)||(S_c \oplus T'))$
$FV \leftarrow S_r, T \leftarrow S_c$
$I_B = I_B \cup (N, A, M, C, FV, T)$
Return (C, FV, T)

On P query on input U

if $\exists V$ *s.t.* $(U, V) \in I_P$ **then**
  return V
else
  $V \xleftarrow{\$} \{0,1\}^b$
  if $\exists U'$ *s.t* $(U', V) \in I_P$ **then**
    bad$\leftarrow$**true** and
    $\boxed{V \xleftarrow{\$} \{0,1\}^b \setminus \{V : (*, V) \in I_P\}}$
  $I_P = I_P \bigcup \{(U, V)\}$
  return V;

Decryption $\mathcal{D}_K(A, C, FV, T)$

$S \leftarrow 0^r||1^c$
if $A \neq \phi$ **then**
  $A_1||A_2 \dots A_a \xleftarrow{r} A$
  for $i = 1$ *to* $a-1$ **do**
    $S \leftarrow P(S_r \oplus A_i||S_c)$
  if $|A_a| == r$ **then**
    $S \leftarrow P((S_r \oplus A_a)||(S_c \oplus 3_c))$
  else
    $S \leftarrow P((S_r \oplus A_a||10^*)||(S_c \oplus 4_c))$
else
  $S \leftarrow P((S_r \oplus 10^{r-1})||(S_c \oplus 4_c))$
$S' \leftarrow P^{-1}(FV||T)$
$N \leftarrow S_r \oplus S'_r$
$T' \leftarrow S_c \oplus S'_c$
$S \leftarrow (0^r \oplus N)||0^c$
$C_1||C_2 \dots C_n \xleftarrow{r} C$
for $i = 1$ *to* $n-1$ **do**
  $M_i \leftarrow S_r \oplus C_i$
  $S \leftarrow P(C_i||S_c)$
if $|C_n| = r$ **then**
  $M_n = C_n \oplus S_r$
  $S \leftarrow S \oplus (M_n||1_c)$
else
  $M_n \leftarrow (S)^{|C_n|} \oplus C_n$
  $S \leftarrow S \oplus ((M_n||10^*)||2_c)$
$S \leftarrow P(S)$
$M \leftarrow M_1||M_2 \dots ||M_n$
$I_B = I_B \cup (N, A, M, C, FV, T)$
Return M

Verification $\mathcal{V}_K(A, C, FV, T)$

Line 1 to 26 same as $\mathcal{D}_K(A, C, FV, T)$
$T'' \leftarrow S_c$
if $(T'' == T')$ **then**
  Return $\top$
else
  Return $\bot$

On P^{-1} **query on input** V

if $\exists U$ *s.t.* $(U, V) \in I_P$ **then**
  return U
else
  $U \xleftarrow{\$} \{0,1\}^b$
  if $\exists V'$ *s.t* $(U, V') \in I_P$ **then**
    bad$\leftarrow$**true** and
    $\boxed{U \xleftarrow{\$} \{0,1\}^b \setminus \{U : (U, *) \in I_P\}}$
  $I_P = I_P \bigcup \{(U, V)\}$
  return U;

Figure 10.5: $\boxed{G_1}$ and G_2: Boxed statements are considered only in game G_1

Initialize $I_B = \phi, I_P = \phi$,$S = S_c||S_r \leftarrow 0^r||0^c$

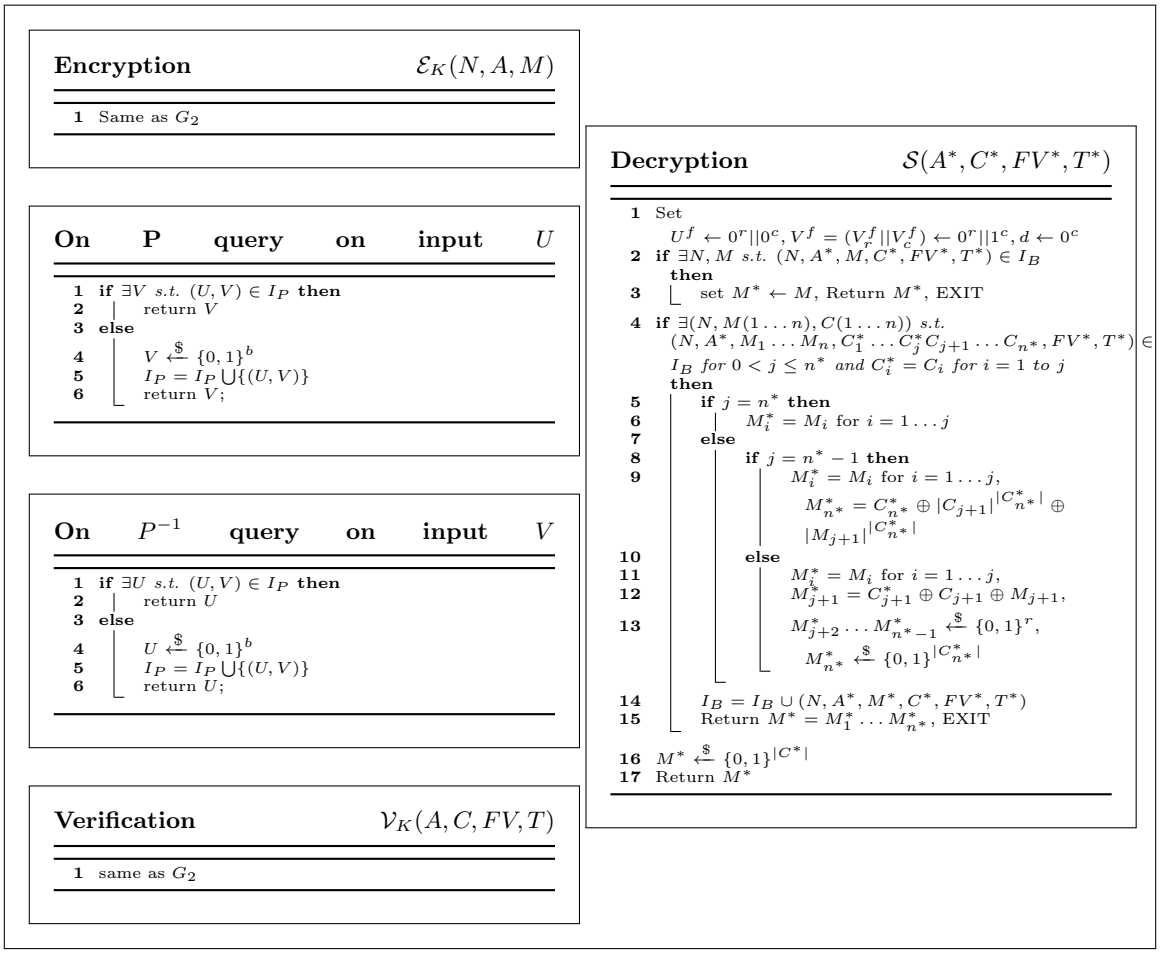


Figure 10.6: G_3

Initialize $I_B = \phi, I_P = \phi$, $S = S_c||S_r \leftarrow 0^r||0^c$, $flag \leftarrow 0$

Encryption $\mathcal{E}_K(N, A, M)$

1 same as G_2

On P query on input U

1 same as G_3

On P^{-1} **query on input** V

1 Same as G_3

Decryption $\mathcal{S}(A^*, C^*, FV^*, T^*)$

1 same as G_3

Verification $\mathcal{V}_K(A, C, FV, T)$

$S \leftarrow 0^r||1^c$
if $A \neq \phi$ **then**
  $A_1||A_2 \dots A_a \xleftarrow{r} A$
  for $i = 1$ *to* $a - 1$ **do**
    if $\exists V$ *s.t.* $(S_r \oplus A_i||S_c, V) \in I_P$ **then**
      $S \leftarrow V$
    else
      $V \xleftarrow{\$} \{0,1\}^b$
      $I_P = I_P \bigcup \{(S_r \oplus A_i||S_c, V)\}$, $S \leftarrow V$
  if $|A_a| == r$ **then**
    if $\exists V$ *s.t.* $((S_r \oplus A_a)||(S_c \oplus 3_c), V) \in I_P$ **then**
      $S \leftarrow V$
    else
      $V \xleftarrow{\$} \{0,1\}^b$
      $I_P = I_P \bigcup \{((S_r \oplus A_a)||(S_c \oplus 3_c), V)\}$, $S \leftarrow V$
  else
    if $\exists V$ *s.t.* $((S_r \oplus A_a||10^*)||(S_c \oplus 4_c), V) \in I_P$ **then**
      $S \leftarrow V$
    else
      $V \xleftarrow{\$} \{0,1\}^b$
      $I_P = I_P \bigcup \{((S_r \oplus A_a||10^*)||(S_c \oplus 4_c), V)\}$, $S \leftarrow V$
else
  if $\exists V$ *s.t.* $((S_r \oplus 10^{r-1})||(S_c \oplus 4_c), V) \in I_P$ **then**
    $S \leftarrow V$
  else
    $V \xleftarrow{\$} \{0,1\}^b$
    $I_P = I_P \bigcup \{((S_r \oplus 10^{r-1})||(S_c \oplus 4_c), V)\}$, $S \leftarrow V$
if $\exists U$ *s.t.* $(U, FV||T) \in I_P$ **then**
  $S' \leftarrow U$
else
  $U \xleftarrow{\$} \{0,1\}^b$
  $I_P = I_P \bigcup \{((U, FV||T)\}$, $S' \leftarrow U$
$N \leftarrow S_r \oplus S'_r$
$T' \leftarrow S_c \oplus S'_c$
$S \leftarrow (0^r \oplus N)||0^c$
$C_1||C_2 \dots C_n \xleftarrow{r} C$
for $i = 1$ *to* $n - 1$ **do**
  $M_i \leftarrow S_r \oplus C_i$
  if $\exists V$ *s.t.* $(C_i||S_c, V) \in I_P$ **then**
    $S \leftarrow V$
  else
    $V \xleftarrow{\$} \{0,1\}^b$
    $I_P = I_P \bigcup \{(C_i||S_c, V)\}$, $S \leftarrow V$
if $|C_n| = r$ **then**
  $M_n = C_n \oplus S_r$
  $S \leftarrow S \oplus (M_n||1_c)$
else
  $M_n \leftarrow (S)^{|C_n|} \oplus C_n$
  $S \leftarrow S \oplus ((M_n||10^*)||2_c)$
if $\exists V$ *s.t.* $(S, V) \in I_P$ **then**
  $S \leftarrow V$
else
  $V \xleftarrow{\$} \{0,1\}^b$
  $I_P = I_P \bigcup \{(S, V)\}$, $S \leftarrow V$
$T'' \leftarrow S_c$
if $(T'' == T')$ **then**
  Return $\top$
else
  Return $\bot$

Figure 10.7: G_4

Initialize $I_B = \phi, I_P = \phi$, $S = S_c||S_r \leftarrow 0^r||0^c$, $flag \leftarrow 0$

Encryption $\mathcal{E}_K(N, A, M)$

1 same as G_2

On P query on input U

1 same as G_3

On P^{-1} **query on input** V

1 Same as G_3

Decryption $\mathcal{S}(A^*, C^*, FV^*, T^*)$

1 same as G_3

Verification $\mathcal{V}_K(A, C, FV, T)$

1 $S \leftarrow 0^r||1^c$
2 **if** $A \neq \phi$ **then**
3 $A_1||A_2 \dots A_a \xleftarrow{r} A$
4 **for** $i = 1$ *to* $a - 1$ **do**
5 **if** $\exists V$ *s.t.* $(S_r \oplus A_i||S_c, V) \in I_P$ **then**
6 $S \leftarrow V$
7 **else**
8 $V \xleftarrow{\$} \{0,1\}^b$, $flag \leftarrow 1$
9 $I_P = I_P \bigcup \{(S_r \oplus A_i||S_c, V)\}$, $S \leftarrow V$
10 **if** $|A_a| == r$ **then**
11 **if** $\exists V$ *s.t.* $((S_r \oplus A_a)||(S_c \oplus 3_c), V) \in I_P$ **then**
12 $S \leftarrow V$
13 **else**
14 $V \xleftarrow{\$} \{0,1\}^b$, $flag \leftarrow 1$
15 $I_P = I_P \bigcup \{((S_r \oplus A_a)||(S_c \oplus 3_c), V)\}$, $S \leftarrow V$
16 **else**
17 **if** $\exists V$ *s.t.* $((S_r \oplus A_a||10^*)||(S_c \oplus 4_c), V) \in I_P$ **then**
18 $S \leftarrow V$
19 **else**
20 $V \xleftarrow{\$} \{0,1\}^b$, $flag \leftarrow 1$
21 $I_P = I_P \bigcup \{((S_r \oplus A_a||10^*)||(S_c \oplus 4_c), V)\}$, $S \leftarrow V$
22 **else**
23 **if** $\exists V$ *s.t.* $((S_r \oplus 10^{r-1})||(S_c \oplus 4_c), V) \in I_P$ **then**
24 $S \leftarrow V$
25 **else**
26 $V \xleftarrow{\$} \{0,1\}^b$, $flag \leftarrow 1$
27 $I_P = I_P \bigcup \{((S_r \oplus 10^{r-1})||(S_c \oplus 4_c), V)\}$, $S \leftarrow V$
28 **if** $\exists U$ *s.t.* $(U, FV||T) \in I_P$ **then**
29 $S' \leftarrow U$
30 **else**
31 $U \xleftarrow{\$} \{0,1\}^b$, $flag \leftarrow 1$
32 $I_P = I_P \bigcup \{((U, FV||T)\}$, $S' \leftarrow U$
33 $N \leftarrow S_r \oplus S'_r$
34 $T' \leftarrow S_c \oplus S'_c$
35 $S \leftarrow (0^r \oplus N)||0^c$
36 $C_1||C_2 \dots C_n \xleftarrow{r} C$
37 **for** $i = 1$ *to* $n - 1$ **do**
38 $M_i \leftarrow S_r \oplus C_i$
39 **if** $\exists V$ *s.t.* $(C_i||S_c, V) \in I_P$ **then**
40 $S \leftarrow V$
41 **else**
42 $V \xleftarrow{\$} \{0,1\}^b$, $flag \leftarrow 1$
43 $I_P = I_P \bigcup \{(C_i||S_c, V)\}$, $S \leftarrow V$
44 **if** $|C_n| = r$ **then**
45 $M_n = C_n \oplus S_r$
46 $S \leftarrow S \oplus (M_n||1_c)$
47 **else**
48 $M_n \leftarrow (S)^{|C_n|} \oplus C_n$
49 $S \leftarrow S \oplus ((M_n||10^*)||2_c)$
50 **if** $\exists V$ *s.t.* $(S, V) \in I_P$ **then**
51 $S \leftarrow V$
52 **else**
53 $V \xleftarrow{\$} \{0,1\}^b$, $flag \leftarrow 1$
54 $I_P = I_P \bigcup \{(S, V)\}$, $S \leftarrow V$
55 $T'' \leftarrow S_c$
56 **if** $(T'' == T')$ **then**
57 **if** $(flag == 1)$ **then**
58 Return $\top$, $bad_3 \leftarrow true$
59 **else**
60 Return $\top$, $bad_4 \leftarrow true$
61 **else**
62 Return $\bot$

Figure 10.8: G_5

Initialize $I_B = \phi, I_P = \phi$, $S = S_c||S_r \leftarrow 0^r||0^c$, $flag \leftarrow 0$

Encryption $\mathcal{E}_K(N, A, M)$

1 same as G_2

On P query on input U

1 same as G_3

On P^{-1} **query on input** V

1 Same as G_3

Decryption $\mathcal{S}(A^*, C^*, FV^*, T^*)$

1 same as G_3

Verification $\mathcal{V}_K(A, C, FV, T)$

1 $S \leftarrow 0^r||1^c$
2 **if** $A \neq \phi$ **then**
3 $A_1||A_2 \dots A_a \xleftarrow{r} A$
4 **for** $i = 1$ *to* $a - 1$ **do**
5 **if** $\exists V$ *s.t.* $(S_r \oplus A_i||S_c, V) \in I_P$ **then**
6 $S \leftarrow V$
7 **else**
8 Return $\perp$, EXIT
9 **if** $|A_a| == r$ **then**
10 **if** $\exists V$ *s.t.* $((S_r \oplus A_a)||(S_c \oplus 3_c), V) \in I_P$ **then**
11 $S \leftarrow V$
12 **else**
13 Return $\perp$, EXIT
14 **else**
15 **if** $\exists V$ *s.t.* $((S_r \oplus A_a||10^*)||(S_c \oplus 4_c), V) \in I_P$ **then**
16 $S \leftarrow V$
17 **else**
18 Return $\perp$, EXIT
19 **else**
20 **if** $\exists V$ *s.t.* $((S_r \oplus 10^{r-1})||(S_c \oplus 4_c), V) \in I_P$ **then**
21 $S \leftarrow V$
22 **else**
23 Return $\perp$, EXIT
24 **if** $\exists U$ *s.t.* $(U, FV||T) \in I_P$ **then**
25 $S' \leftarrow U$
26 **else**
27 Return $\perp$, EXIT
28 $N \leftarrow S_r \oplus S'_r$
29 $T' \leftarrow S_c \oplus S'_c$
30 $S \leftarrow (0^r \oplus N)||0^c$
31 $C_1||C_2 \dots C_n \xleftarrow{r} C$
32 **for** $i = 1$ *to* $n - 1$ **do**
33 $M_i \leftarrow S_r \oplus C_i$
34 **if** $\exists V$ *s.t.* $(C_i||S_c, V) \in I_P$ **then**
35 $S \leftarrow V$
36 **else**
37 Return $\perp$, EXIT
38 **if** $|C_n| = r$ **then**
39 $M_n = C_n \oplus S_r$
40 $S \leftarrow S \oplus (M_n||1_c)$
41 **else**
42 $M_n \leftarrow (S)^{|C_n|} \oplus C_n$
43 $S \leftarrow S \oplus ((M_n||10^*)||2_c)$
44 **if** $\exists V$ *s.t.* $(S, V) \in I_P$ **then**
45 $S \leftarrow V$
46 **else**
47 Return $\perp$, EXIT
48 $T'' \leftarrow S_c$
49 Return $\perp$

Figure 10.9: G_6

Initialize $I_B = \phi, I_P = \phi$, $S = S_c||S_r \leftarrow 0^r||0^c$

Encryption $\mathcal{E}_K(N, A, M)$

1 $S \leftarrow (0^r \oplus N)||0^c$
2 $S \leftarrow P(S)$
3 **if** $M \neq \phi$ **then**
4 $\quad M_1||M_2 \dots M_n \xleftarrow{r} M$
5 $\quad$ **for** $i = 1$ *to* $n-1$ **do**
6 $\quad\quad C_i \leftarrow S_r \oplus M_i$
7 $\quad\quad S \leftarrow P(C_i||S_c)$
8 $\quad$ **if** $|M_n| = r$ **then**
9 $\quad\quad S \leftarrow S \oplus (M_n||1_c)$
10 $\quad\quad C_n \leftarrow S_r$
11 $\quad$ **else**
12 $\quad\quad S \leftarrow S \oplus ((M_n||10^*)||2_c)$
13 $\quad\quad C_n \leftarrow (S)^{|M_n|}$
14 $\quad S \leftarrow P(S)$
15 $\quad C \leftarrow \{0,1\}^{|M|}$
16 $\quad \boxed{C \leftarrow C_1||C_2 \dots ||C_n}$
17 **else**
18 $\quad S \leftarrow P((S_r \oplus 10^{r-1})||(S_c \oplus 2_c))$
19 $\quad C \leftarrow \phi$
20 $T' \leftarrow S_c$
21 $S \leftarrow 0^r||1^c$
22 **if** $A \neq \phi$ **then**
23 $\quad A_1||A_2 \dots A_a \xleftarrow{r} A$
24 $\quad$ **for** $i = 1$ *to* $a-1$ **do**
25 $\quad\quad S \leftarrow P(S_r \oplus A_i||S_c)$
26 $\quad$ **if** $|A_a| == r$ **then**
27 $\quad\quad S \leftarrow P((S_r \oplus A_a)||(S_c \oplus 3_c))$
28 $\quad$ **else**
29 $\quad\quad S \leftarrow P((S_r \oplus A_a||10^*)||(S_c \oplus 4_c))$
30 **else**
31 $\quad S \leftarrow P((S_r \oplus 10^{r-1})||(S_c \oplus 4_c))$
32 $S \leftarrow P((S_r \oplus N)||(S_c \oplus T'))$
33 $FV \xleftarrow{\$} \{0,1\}^r, T \xleftarrow{\$} \{0,1\}^c$
34 $\boxed{FV \leftarrow S_r, T \leftarrow S_c}$
35 $I_B = I_B \cup (N, A, M, C, FV, T)$
36 Return (C, FV, T)

Decryption $\mathcal{S}(A^*, C^*, FV^*, T^*)$

1 same as G_3

Verification $\mathcal{V}_K(A^*, C^*, FV^*, T^*)$

1 Return $\perp$

On P query on input U

1 same as G_3

On P^{-1} **query on input** V

1 Same as G_3

Figure 10.10: $\boxed{G_7}$ and G_8: Boxed statements are considered only in game G_7

10.7.2 Limitation

In the above proof, we have provided a security of RSAEB upto $c/2$ bits, which is less than the security provided by the SAEB scheme. As a future work, we can work on improving the security bound of RSAEB by using a multi-collision concept.

10.8 Conclusion

In this chapter, we have explained a PA1 attack on SAEB authenticated encryption scheme followed by proposing a new AE scheme called RSAEB, which prevent this attack. We have provided a security proof for this newly proposed AE scheme in AERUP model.

Chapter 11

Conclusion & Future Work

In this chapter, we summarize our result presented in this thesis and we attempt to identify research directions on which we will work in future.

11.1 Summary

In this thesis, we focused on the designing of authenticated encryption schemes suitable for memory constrained devices.

1. sp-AELM is a sponge based authenticated encryption scheme that provides support for memory constrained devices. We also provide its security proof for privacy and authenticity in an ideal permutation model, using a code based game playing framework. Furthermore, we also present two more variants of sp-AELM that serve the same purpose and are more efficient than sp-AELM.

2. We also analyzed sponge based CAESAR submissions using our proposed technique, to determine their potential to support limited memory constraint.

3. dAELM is a deterministic authenticated encryption scheme providing support for memory constrained device. Deterministic AE (DAE) is used in domains such as the key wrap, where the available message entropy omits the overhead of a nonce. For limiting memory usage, our idea is to use a session key to encrypt a message and share the session key with the user depending upon the verification of a tag. We provide the security proof of the proposed construction in the ideal cipher model.

4. We have shown a PA1 attack on recently proposed lightweight authenticated encryption scheme SAEB [60]. Then we have proposed a modification to SAEB to overcome this attack. We provided two modified versions of SAEB called RSAEB : first one provides PA1 security in nonce respecting scenario and another one provides PA1 security in nonce misuse scenario and PA2 security. PA2 is stronger security notion than PA1, which comes at the cost of an additional pass.

11.2 Future Work

In this section, we attempt to identify future research directions in which we can try applying the topics discussed in this thesis.

1. Recently in [5], authors presented a generalized technique to add a RUP security to general class of encryption schemes. Their scheme implicitly assumes the unlimited memory scenario. However, it turns out to be insecure when implemented in memory constrained devices. Hence, improving the security of this scheme in limited memory scenario is one of the potential future work.

2. We can generalize the concept of cryptographic module and proposed a new security notion for authenticated encryption schemes based on the cryptographic module approach.

3. In chapter 10, we presented a new authenticated encryption scheme RSAEB which is a RUP secure variant of SAEB. However, this new variant requires an inverse operation, which makes the scheme not practical in a lightweight environment. In this work, we have just laid the foundation for the RUP secure variant of SAEB. We have planned to continue this work and coming up with a more efficient and inverse-free variant of RSAEB.

Bibliography

[1] Fips pub 140-2, security requirements for cryptographic modules. Federal Information Processing Standards Publication (Supercedes FIPS PUB 140-1, 1994 January 11), 2001. http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf.

[2] CAESAR: Competition for authenticated encryption: Security, applicability, and robustness, 2014. `http://competitions.cr.yp.to/caesar.html`.

[3] Megha Agrawal, Donghoon Chang, and Somitra Sanadhya. sp-AELM: Sponge based authenticated encryption scheme for memory constrained devices. In Ernest Foo and Douglas Stebila, editors, *Information Security and Privacy - 20th Australasian Conference, ACISP 2015, Brisbane, QLD, Australia, June 29 - July 1, 2015, Proceedings*, volume 9144 of *Lecture Notes in Computer Science*, pages 451–468. Springer, 2015.

[4] Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. APE: Authenticated permutation-based encryption for lightweight cryptography. *IACR Cryptology ePrint Archive*, 2013:791, 2013.

[5] Tomer Ashur, Orr Dunkelman, and Atul Luykx. Boosting authenticated encryption robustness with minimal modifications. In *Annual International Cryptology Conference*, pages 3–33. Springer, 2017.

[6] Subhadeep Banik, Andrey Bogdanov, Atul Luykx, and Elmar Tischhauser. SUNDAE: Small Universal Deterministic Authenticated Encryption for the Internet of Things. *IACR Transactions on Symmetric Cryptology*, pages 1–35, 2018.

[7] Guy Barwell, Daniel Page, and Martijn Stam. Rogue decryption failures: Reconciling ae robustness notions. In Jens Groth, editor, *Cryptography and Coding*, pages 94–111, Cham, 2015. Springer International Publishing.

[8] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *Annual International Cryptology Conference*, pages 1–15. Springer, 1996.

[9] Mihir Bellare and Chanathip Namprempre. Authenticated Encryption: Relations Among Notions and Analysis of the Generic Composition Paradigm. *J. Cryptol.*, 21(4):469–491, September 2008.

[10] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.

[11] Mihir Bellare and Phillip Rogaway. Encode-Then-Encipher Encryption: How to Exploit Nonces or Redundancy in Plaintexts for Efficient Cryptography. In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 317–330. Springer, 2000.

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

[13] Mihir Bellare, Phillip Rogaway, and David Wagner. EAX: A Conventional Authenticated-Encryption Mode. *IACR Cryptology ePrint Archive*, 2003:69, 2003.

[14] 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.

[15] Guido Bertoni, Joan Daemen, Michal Peeters, and Gilles Van Assche. Cryptographic sponge functions, 2011. `http://sponge.noekeon.org/`.

[16] Bertoni, Guido and Daemen, Joan and Peeters, Michaël and Van Assche, Gilles. Duplexing the Sponge: Single-pass Authenticated Encryption and Other Applications. In *Proceedings of the 18th International Conference on Selected Areas in Cryptography*, SAC'11, pages 320–337, 2012.

[17] Arghya Bhattarcharjee, Eik List, Cuauthemoc Mancillas-Lopéz, and Mridul Nandi. Security proofs for oribatida. 2019.

[18] Mittal S Bhiogade. Secure socket layer. In *Computer Science and Information Technology Education Conference*, pages 85–90, 2002.

[19] John Black. The ideal-cipher model, revisited: An uninstantiable blockcipher-based hash function. In *International Workshop on Fast Software Encryption*, pages 328–340. Springer, 2006.

[20] Matt Blaze. High-Bandwidth Encryption with Low-Bandwidth Smartcards. In Dieter Gollmann, editor, *FSE*, volume 1039 of *Lecture Notes in Computer Science*, pages 33–40. Springer, 1996.

[21] Matt Blaze, Joan Feigenbaum, and Moni Naor. A Formal Treatment of Remotely Keyed Encryption. In Kaisa Nyberg, editor, *EUROCRYPT*, volume 1403 of *Lecture Notes in Computer Science*, pages 251–265. Springer, 1998.

[22] Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting mobile communications: the insecurity of 802.11. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 180–189. ACM, 2001.

[23] Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, and Mridul Nandi. Blockcipher-based Authenticated Encryption: How Small Can We Go? In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 277–298. Springer, 2017.

[24] Donghoon Chang, Nilanjan Datta, Avijit Dutta, Bart Mennink, Mridul Nandi, Somitra Sanadhya, and Ferdinand Sibleyras. Release of unverified plaintext: Tight unified model and application to anydae. Cryptology ePrint Archive, Report 2019/1326, 2019. `https://eprint.iacr.org/2019/1326`.

[25] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, Martin Schlaffer. Ascon v1. `http://competitions.cr.yp.to/round1/asconv1.pdf`.

[26] Don Coppersmith. The data encryption standard (des) and its strength against attacks. *IBM journal of research and development*, 38(3):243–250, 1994.

[27] J Daemen and V Rijmen. Fips pub 197: Advanced encryption standard (aes). *National Institute of Standards and Technology (NIST)*, 2001.

[28] Joan Daemen and Vincent Rijmen. The rijndael block cipher: Aes proposal. In *First candidate conference (AeS1)*, pages 343–348, 1999.

[29] Danilo Gligoroski, Hristina Mihajloska, Simona Samardjiska, Hakon Jacobsen, Mohamed El-Hadedy and Rune Erlend Jensen. PiCipher v1. `http://competitions.cr.yp.to/round1/picipherv1.pdf`.

[30] Tim Dierks and Eric Rescorla. The transport layer security (tls) protocol version 1.2. Technical report, 2008.

[31] Whitfield Diffie and Martin E Hellman. Special feature exhaustive cryptanalysis of the nbs data encryption standard. *Computer*, 10(6):74–84, 1977.

[32] Yevgeniy Dodis. Concealment and Its Applications to Authenticated Encryption. In Alexander W. Dent and Yuliang Zheng, editors, *Practical Signcryption*, Information Security and Cryptography, pages 149–173. Springer, 2010.

[33] Morris J. Dworkin. Sp 800-38c. recommendation for block cipher modes of operation: The CCM mode for authentication and confidentiality. Technical report, Gaithersburg, MD, United States, 2004.

[34] D. Eastlake, 3rd and P. Jones. US Secure Hash Algorithm 1 (SHA1), 2001.

[35] Elena Andreeva and Andrey Bogdanov and Atul Luykx and Bart Mennink and 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.

[36] Elena Andreeva, Begul Bilgin, Andrey Bogdanov, Atul Luykx, Florian Mendel , Bart Mennink, Nicky Mouha, Qingju Wang and Kan Yasuda . PRIMATEs v1. `http://competitions.cr.yp.to/round1/primatesv1.pdf`.

[37] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext transfer protocol–http/1.1. Technical report, 1999.

[38] Pierre-Alain Fouque, Antoine Joux, Gwenaëlle Martinet, and Frédéric Valette. Authenticated On-Line Encryption. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 145–159. Springer, 2003.

[39] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 365–377. ACM, 1982.

[40] Guido Bertoni and Joan Daemen and Michal Peeters and Gilles Van Assche. G.V.: Permutationbased encryption, authentication and authenticated encryption, 2012.

[41] Guido Bertoni, Joan Daemen, Michael Peeters, Gilles Van Assche, Ronny Van Keer. Ketje v1. `http://competitions.cr.yp.to/round1/ketjev11.pdf`.

[42] Guido Bertoni, Joan Daemen, Michael Peeters, Gilles Van Assche, Ronny Van Keer. Ketje v1. `http://keyak.noekeon.org/Keyak-1.2.pdf`.

[43] Martin E Hellman. Des will be totally insecure within ten years. *IEEE spectrum*, 16(7):32–40, 1979.

[44] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. Robust authenticated-encryption aez and the problem that it solves. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 15–44. Springer, 2015.

[45] 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*, pages 493–517, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[46] Paul Hoffman. Cryptographic suites for ipsec. Technical report, 2005.

[47] Hongjun Wu, Tao Huang. JAMBU Lightweight Authenticated Encryption Mode and AES-JAMBU (v1), 2014. http://competitions.cr.yp.to/round1/aesjambuv1.pdf.

[48] Tetsu Iwata and Kaoru Kurosawa. Omac: One-key cbc mac. In *International Workshop on Fast Software Encryption*, pages 129–153. Springer, 2003.

[49] Javad Alizadeh, Mohammad Reza Aref and Nasour Bagheri. Artemia v1. http://competitions.cr.yp.to/round1/artemiav1.pdf.

[50] Samuel Neves Jean-Philippe Aumasson, Philipp Jovanovic. NORX: Parallel and Scalable AEAD, 2014. https://norx.io/.

[51] Ari Juels. Minimalist cryptography for low-cost rfid tags. In *International conference on security in communication networks*, pages 149–164. Springer, 2004.

[52] Charanjit S. Jutla. Encryption modes with almost free message integrity. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, volume 2045 of *Lecture Notes in Computer Science*, pages 529–544. Springer, 2001.

[53] Steven L Kinney. *Trusted platform module basics: using TPM in embedded systems.* Elsevier, 2006.

[54] Tadayoshi Kohno, John Viega, and Doug Whiting. CWC: A High-Performance Conventional Authenticated Encryption Mode. In Bimal K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 408–426. Springer, 2004.

[55] Stefan Lucks. On the Security of Remotely Keyed Encryption. In Eli Biham, editor, *FSE*, volume 1267 of *Lecture Notes in Computer Science*, pages 219–229. Springer, 1997.

[56] Markku-Juhani O. Saarinen. The CBEAMr1 Authenticated Encryption Algorithm. http://competitions.cr.yp.to/round1/cbeamr1.pdf.

[57] Markku-Juhani O. Saarinen. The STRIBOBr1 Authenticated Encryption Algorithm. http://competitions.cr.yp.to/round1/stribobr1.pdf.

[58] David A. McGrew and John Viega. The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In Anne Canteaut and Kapalee Viswanathan, editors, *INDOCRYPT*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2004.

[59] Kazuhiko Minematsu. Parallelizable rate-1 authenticated encryption from pseudorandom functions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 275–292. Springer, 2014.

[60] Yusuke Naito, Mitsuru Matsui, Takeshi Sugawara, and Daisuke Suzuki. SAEB: A Lightweight Blockcipher-Based AEAD Mode of Operation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 192–217, 2018.

[61] Jacques Patarin. The ?coefficients h? technique. In *International Workshop on Selected Areas in Cryptography*, pages 328–345. Springer, 2008.

[62] Pawel Morawiecki, Kris Gaj, Ekawat Homsirikamol, Krystian Matusiewicz, Josef Pieprzyk, Marcin Rogawski, Marian Srebrny, and Marcin Wojcik. ICEPOLE v1. `http://competitions.cr.yp.to/round1/icepolev1.pdf`.

[63] Jon Postel and Joyce Reynolds. File transfer protocol. Technical report, 1985.

[64] Jon Postel and Joyce K Reynolds. Telnet protocol specification. Technical report, 1983.

[65] R. Rivest. The MD5 Message-Digest Algorithm, 1992.

[66] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 98–107. ACM, 2002.

[67] Phillip Rogaway, Mihir Bellare, and John Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3):365–403, 2003.

[68] Phillip Rogaway and Thomas Shrimpton. Deterministic authenticated-encryption: A provable-security treatment of the key-wrap problem. *IACR Cryptology ePrint Archive*, 2006:221, 2006.

[69] Phillip Rogaway and David A Wagner. A critique of ccm. *IACR Cryptology ePrint Archive*, 2003:70, 2003.

[70] Bruce Schneier. *Applied cryptography: protocols, algorithms, and source code in C*. john wiley & sons, 2007.

[71] Claude Elwood Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

[72] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs, 2004. *URL: http://eprint. iacr. org/2004/332*, 2006.

[73] Petr Švenda. Basic comparison of modes for authenticated-encryption (iapm, xcbc, ocb, ccm, eax, cwc, gcm, pcfb, cs), 2016.

[74] Tetsu Iwata and Kazuhiko Minematsu and Jian Guo and Sumio Morioka . CLOC: Compact Low-Overhead CFB, 2014. `http://competitions.cr.yp.to/round1/clocv1.pdf`.

[75] Tetsu Iwata and Kazuhiko Minematsu and Jian Guo and Sumio Morioka and Eita Kobayashi. SILC: SImple Lightweight CFB, 2015. `https://competitions.cr.yp.to/round2/silcv2.pdf`.

[76] Erik Tews, Ralf-Philipp Weinmann, and Andrei Pyshkin. Breaking 104 bit wep in less than 60 seconds. In *International Workshop on Information Security Applications*, pages 188–202. Springer, 2007.

[77] Serge Vaudenay. Security Flaws Induced by CBC Padding Applications to SSL, IPSEC, WTLS... In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 534–545. Springer, 2002.

[78] Hongjun Wu and Bart Preneel. AEGIS: A Fast Authenticated Encryption Algorithm. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography*, volume 8282 of *Lecture Notes in Computer Science*, pages 185–201. Springer, 2013.

[79] Tatu Ylonen. Ssh-secure login connections over the internet. In *Proceedings of the 6th USENIX Security Symposium*, volume 37, 1996.

[80] Tatu Ylonen and Chris Lonvick. The secure shell (ssh) transport layer protocol. Technical report, 2005.