On Message Authentication in 4G LTE System

by

Teng Wu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2015

Abstract

After decades of evolution, the cellular system has become an indispensable part of modern life. Together with the convenience brought by the cellular system, many security issues have arisen. Message integrity protection is one of the urgent problems. The integrity of a message is usually protected by message authentication code (MAC).

Forgery attacks are the primary threat to message integrity. By Simon's definition, forgery is twofold. The first is impersonation forgery, in which the opponent can forge a MAC without knowing any message-MAC pairs. The second is substitution forgery, in which the opponent can forge a MAC by knowing certain message-MAC pairs.

In the 4G LTE system, MAC is applied not only to RRC control messages and user data, but also to authentication of the identities in the radio network during the authentication and key agreement (AKA) procedure. There is a set of functions used in AKA, which is called A3/A8. Originally, only one cipher suite called MILENAGE followed the definition of A3/A8. Recently, Vodafone has proposed another candidate called TUAK.

This thesis first analyzes a MAC algorithm of the 4G LTE system called EIA1. The analysis shows that because of its linear structure, given two valid message-MAC pairs generated by EIA1, attackers can forge up to 2^{32} valid MACs by the algorithm called linear forgery attack proposed in this thesis. This thesis also proposes a well-designed scenario, in which attackers can apply the linear forgery attack to the real system.

The second work presented in this thesis fixes the gap between the almost XOR universal property and the substitution forgery probability, and assesses the security of EIA1 under different attack models. After the security analysis, an optimized EIA1 using an efficient polynomial evaluation method is proposed. This polynomial evaluation method is analog to the fast Fourier transform. Compared with Horner's rule, which is used in the official implementation of EIA1, this method reduces the number of multiplications over finite field dramatically. The improvement is shown by the experiment results, which suggests that the optimized code is much faster than the official implementation, and the polynomial evaluation method is better than Horner's rule.

The third work in this thesis assesses the security of TUAK, and proves TUAK is a secure algorithm set, which means f_1, f_1^*, and f_2 are resistant to forgery attacks, and key recovery attacks; f_3 - f_5, and f_5^* are resistant to key recovery attacks and collision. A novel technique called multi-output filtering model is proposed in this work in order to study the non-randomness property of TUAK and other cryptographic primitives, such as AES, KASUMI, and PRESENT. A multi-output filtering model consists of a linear feedback shift register (LFSR) and a multi-output filtering function. The contribution of this research is twofold. First, an attack technique under IND-CPA using the multi-output filtering model is proposed. By introducing a distinguishing function, we theoretically determine the success rate of this attack. In particular, we construct a distinguishing function based on the distribution of the linear complexity of component sequences, and apply it on studying TUAK's f_1 algorithm, AES, KASUMI and PRESENT. The experiments demonstrate that the success rate of the attack on KASUMI and PRESENT is non-negligible, but f_1 and AES are resistant to this attack. Second, this research studies the distribution of the cryptographic properties of component functions of a random primitive in the multi-output filtering model. The experiments show some non-randomness in the distribution of the algebraic degree and nonlinearity for KASUMI.

The last work is constructing two MACs. The first MAC called WGIA-128 is a variant of EIA1, and requires the underlying stream cipher to generate uniform distributed key streams. WG-16, a stream cipher with provable security, is a good choice to be the underlying cipher of WGIA-128 because it satisfies the requirement. The second MAC called AMAC is constructed upon APN functions. we propose two different constructions of AMAC, and both of these two constructions have provable security. The probability of substitution forgery attacks against both constructions of AMAC is upper bounded by a negligible value. Compared with EIA1 and EIA3, two message authentication codes used in the 4G LTE system, both constructions of AMAC are slower than EIA3, but much faster than EIA1. Moreover, both constructions of AMAC are resistant to cycling and linear forgery attacks, which can be applied to both EIA1 and EIA3.

Acknowledgements

First and foremost, I would like to extend my immeasurable appreciation and deepest gratitude towards my advisor Professor Guang Gong for her instructive guidance and tremendous support during the past four years. Her critical thinking and positive attitude towards work and life have always inspired and motivated me, and her extensive knowledge and experience have been the biggest help and resource for my studies. The last four years have been an enjoyable and unforgettable time that has given me great academic and personal development, thanks to the role model that Professor Gong has provided as a successful person, researcher, and professor.

I would also like to express my sincerest appreciation to Professor Rei Safavi-Naini from the University of Calgary for serving as my external examiner and providing many valuable suggestions. My deepest gratitude also goes to my thesis examining committee members, Professor David Jao, Professor Kshirasagar Naik, and Professor Mahesh Tripunitara, for their instrumental comments and their time on this thesis. I am deeply honored and blessed to have been guided by them. The thesis would not have been possible without their assistance.

I am particularly grateful for the knowledge, inspiration, and encouragement that I received from Professor Dong Zheng, Professor Kefei Chen and Professor Xuejia Lai at Shanghai Jiao Tong University during my Master's study. It was they who motivated me and led me to pursue my interests in cryptography and security.

I am deeply obliged to many of my friends and colleagues at the University of Waterloo for their enormous support and inspiring suggestions during and beyond my PhD program: Professor Mark Aagaard, Dr. Xinxin Fan, Dr. Fei Huo, Dr. Bo Zhu, Dr. Kalikinkar Mandal, Dr. Yin Tan, Dr. Valentin Suder, Qiao Liu, Yao Chen, Gangqiang Yang, Shasha Zhu, Nusa Zidaric and Kaveh Fazli. I want to thank all the wonderful members of the Communication Security (ComSec) Lab of the University of Waterloo for always contributing to create a pleasant and stimulating group atmosphere.

Last but not least, I want to thank my family for their unconditional and endless love, trust, support and understanding. Thanks go to my father and mother for supporting and advising me at every important time of my life. Thanks are due to my wife for her thoughtful care that encourages me to adventure in the academic world. I also owe a special debt of gratitude to my daughter for the happiness brought by her. I would not be where I am today without my lovely family.

Dedication

I would like to dedicate my thesis
to my wife Ms. Luchen Tan,
my parent Mr. Hongju Wu and Ms. Shuqing Zhang,
and my daughter Tanya Wu.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the past thirty years, wireless communication has acquired tremendous success. Now it is playing a key role in every corner of the human society. Cellular and Wi-Fi are two widely deployed wireless communication systems. Because of their outstanding mobility and flexibility, they offer great convenience to our lives.

The cellular system has already experienced four evolutions. The connection speed and the management method have been substantially improved. Compared with the cellular system, Wi-Fi is designed to cover much smaller areas. Thus, it is simpler than the cellular system. The Wi-Fi alliance concentrates on improving the coverage of a single access point (AP) and the speed of the network.

Together with its convenience, wireless communication also gives rise to security concerns. Since the data are transmitted over the air, the signal can be intercepted or modified easily by an attacker. Moreover, the expense of attacking a wireless communication system has decreased to an affordable level. Therefore, the vulnerability of a wireless communication system is exposed to everyone. Protecting wireless communication systems has become an urgent problem to be solved. In this thesis, we focus on the integrity protection of the 4G LTE system. We remark that some methods can be extended to other wireless communication systems as well. But we will not discuss the extension in this thesis.

The rest of this chapter is organized as follows. Section 1.1 introduces the brief history of the Cellular system. Section 1.2 demonstrates the authentication and message authentication in the 4G LTE system. Some well-known message authentication code constructions are presented in Section 1.3. The last section, Section 1.6, illustrates the structure of this thesis.

1.1 Historical View of Cellular Network

Amazingly, the first “wireless” conversation happened in 1880. It was made by Alexander Graham Bell and Charles Sumner Tainter. This communication device was called “Photophone”. As what it literally meant, the communication was conducted via light, and the communication distance could not be longer than the sight distance. Although the photophone was hardly practical in that era, as a precursor, it in fact started the age of wireless communications.

The concept of our daily used cellphone was proposed by AT&T’s BELL lab in the 1970’s, more than one hundred years after Heinrich Hertz showed the existence of electromagnetic waves. In 1981, the first commercial cellphone network was deployed by Nordic Mobile telephone, which is the predecessor of Nokia. Two years later, the US also had its own commercial cellphone service called Advanced Mobile Phone Service. This generation of cellphone is usually referred to as the 1G network. The communication of the 1G network was conducted using analog signal. To serve multiple users simultaneously, the 1G network used Frequency Division Multiple Access (FDMA) to divide the channel.

In 1982, Europe began to work on the next generation communication technology, which was envisioned to be a digital cellular communication network. Five years later, the specification of the Global System for Mobile communications (GSM) was approved by the European countries and became the standard in Europe. The first phone call via GSM network was made in 1991, and in the next year, the first text message was sent through the GSM network. GSM is a digital communication network. It divides the channel by Time Division Multiple Access (TDMA). Besides GSM, there is another standard of the 2G network, which is called CDMA1 (IS-95 or TIA-EIA-95). CDMA1 was standardized by the US. As the commercial name suggests, it divides the channel by Code Division Multiple Access (CDMA).

At the beginning of GSM, it only allowed to transmit data through a circuit switched fashion. Thus, the data were charged per minute of connection time. As the General Packet Radio Service (GPRS) was developed, GSM had the capability to transmit data in a packet switched way, and the data was billed based on the volume of data transmitted. This GSM/GPRS network is usually called the 2.5G network. The difference between GSM and GSM/GPRS is that GSM/GPRS imported packet switching into the cellular system.

GSM and CDMA1 were two mainstream standards in the age of the 2G network. As the competition of 3G network started, GSM and CDMA1 evolved to UMTS and CDMA2000 respectively. UMTS was standardized by an organization in Europe called 3GPP, and CDMA2000 was operated by a group in North America called 3GPP2. UMTS is based on

WCDMA, which is also a variant of CDMA. In the 3G network, all data is transmitted via packet switched network. However, the voice is still transmitted through the circuit switched network.

The competition between different standards and organizations were terminated by the rise of 4G LTE. Now, LTE has become the only standard of the 4G network. It evolved from UMTS. A new technology called Orthogonal Frequency Division Multiplexing (OFDM) was introduced in the LTE standard. Both data and voice should be transmitted via pack switched network. However, because of the legacy infrastructure in some network, the voice phone call is made by downgrading the network to 3G or 2G, which means the voice is transmitted by a circuit switched method in those systems. This downgrading brought some security issues to the LTE system.

1.2 Authentication and Key Agreement Procedure and Message Authentication in 4G LTE System

There are two kinds of message authentications in the 4G LTE system: one used in the Authentication and Key Agreement (AKA) procedure and the other applied to RRC commands and the user data.

The AKA procedure is presented in Figure 1.1 [8, 38]. The cellphone is called User Equipment (UE) in the 4G LTE standard. The Subscriber Identity Module (SIM) is called USIM in both 3G UMTS and 4G LTE. MME is the Mobility Management Entity, which is called RNC in 3G UMTS. AuC is the Authentication Centre. K is the long term credential, which resides only in USIM and AuC. Note that K is written in a secure ROM in USIM, and cannot be read out or be changed. USIM provides a computation interface for the cellphone using but not outputting the long term key. The interface is called A3/A8 algorithm. Although it is called an algorithm, in fact, it only specifies the input and output formats. The underlying cipher is chosen by the operator. In 2G GSM, because of the weakness of the first version of A3/A8, the long term key was exposed easily, and many SIM card cloning tools were seen in the market at that time. However, the evolution of the underlying cipher led to the current version of A3/A8, which is considered to be secure.

The AKA procedure starts with sending the International Mobile Subscriber Identity (IMSI) by UEs to the MME. The IMSI is an unique number, which associates the USIM card with an account. The MME forwards the IMSI to AuC. After AuC gets the identity, it retrieves the long term key K and generates the Authentication Vectors (AVs) using A3/A8. AuC generates several AVs and sends them all to the MME, which selects one

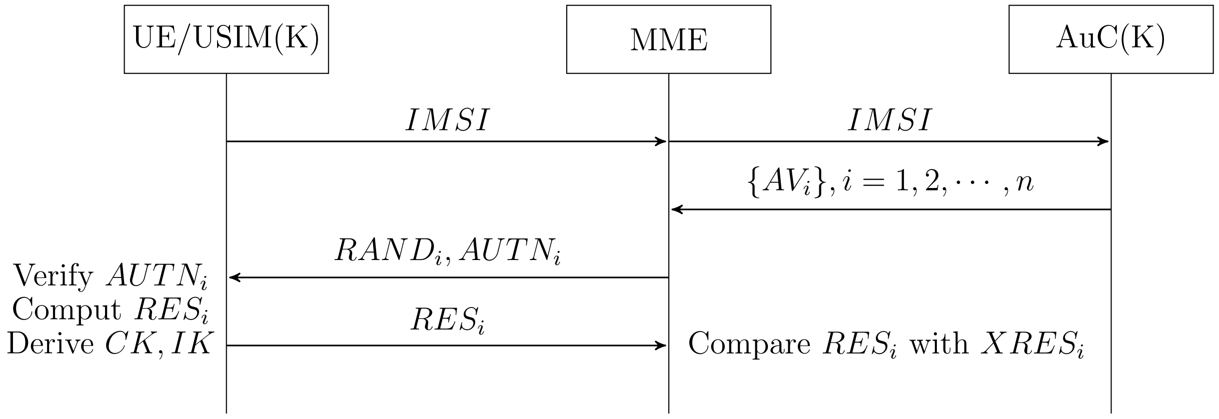


Figure 1.1: The AKA procedure of 4G LTE

AV each time to authenticate the UE. Before using up all the AVs sent by AuC, the MME can carry out the AKA for the same USIM by itself without involving the AuC again. This mechanism saves the bandwidth between the MME and AuC. This feature is important, because, especially when UE is roaming, the communication between the local MME and AuC is expensive. MME retrieves the random number $RAND_i$ and the authentication token $AUTN_i$ from the i-th AV, and sends them back to the UE. Upon receiving the response from the MME, the UE verifies $AUTN_i$ to authenticate the received sequence number SQN and computes a response, which will be sent back to the MME. The Confidentiality Key (CK) and Integrity Key (IK) are also derived in this process. After the response RES_i is sent back to MME, MME authenticates UE by comparing RES_i with $XRES_i$, which is computed by AuC and written in AV_i.

In 3G UMTS, AV has five fields,

$$AV_i = \{RAND_i, XRES_i, CK_i, IK_i, AUTN_i\}.$$

As described above, $RAND_i$ is a random number. $XRES_i$ is the expected response. CK_i and IK_i are the confidentiality key and integrity key respectively. (In 4G LTE, these two keys are replaced by one master session key K_{ASME}.) $AUTN_i$ is the authentication token, which is defined by

$$AUTN_i = \{SQN_i \oplus AK_i, AMF, MAC_i\}.$$

AK is the anonymity key, which masks the sequence number SQN_i. Note that SQN_i is a relatively sensitive data, by which an intruder can infer some information about the user, since the user utilizes it for the verification of the authenticity of the network. Thus, it

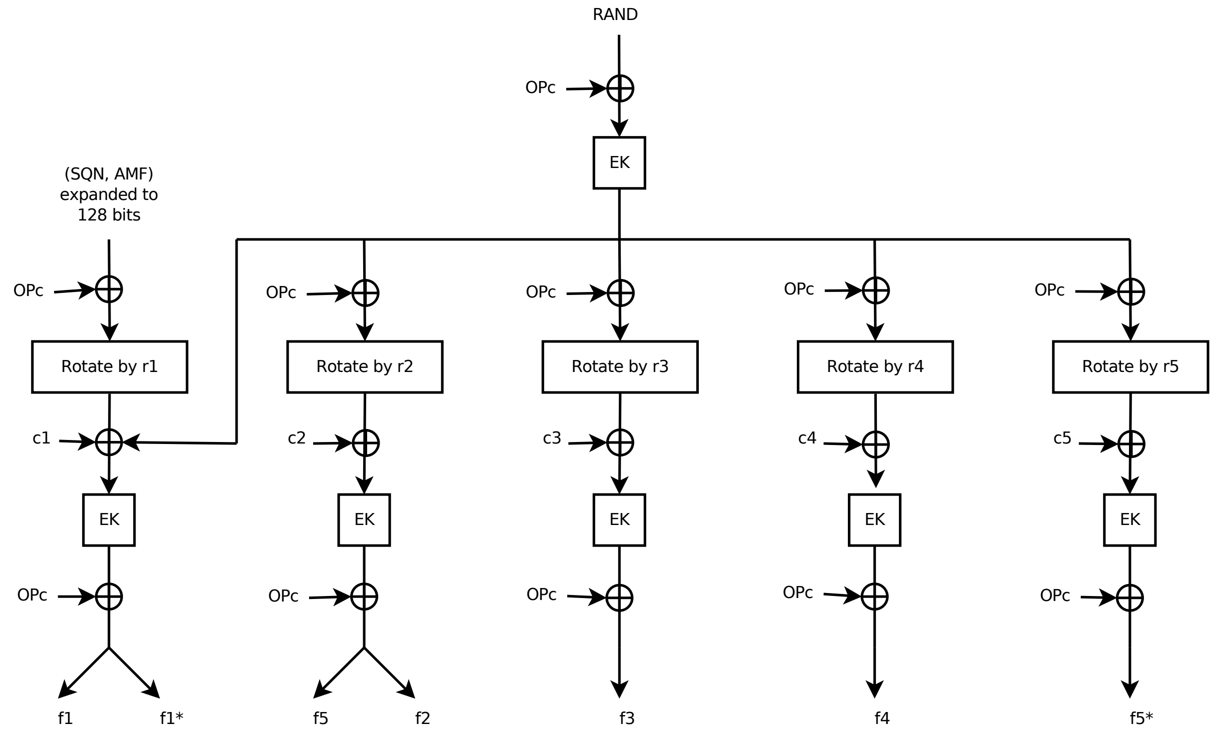


Figure 1.2: MILENAGE

is masked by a random number. AMF is the authentication management field, which indicates some parameters of the authentication. MAC_i is the Message Authentication Code (MAC) computed over the information in $AUTN_i$. To verify $AUTN_i$, UE computes MAC_i' and compares it with MAC_i.

Together, all the message authentication and key derivation functions mentioned above make up the A3/A8 algorithm. Actually, A3/A8 algorithm contains seven functions. Originally, only one algorithm set called MILENAGE followed the A3/A8 standard. Recently, Vodafone proposed another one called TUAK. Since TUAK is one of the research topic of this thesis, it is introduced in the next section together with other preliminaries of this thesis. In this section, we use MILENAGE as an example to illustrate the A3/A8 algorithm.

MILENAGE is presented in Figure 1.2. The c_i and r_i, for $1 \leq i \leq 5$, are constants. OP_c is an operator specified constant. E_k is an arbitrary block cipher, whose block is 128-bit. Usually, the network operators choose AES as the underlying block cipher. The function f_1 generates the MAC of $AUTN$. In the specification, the f_1 function is called the network authentication function, which means the UE uses this function to authenticate the network. The f_2 function, called the user authentication function, generates the expected response and response. The f_3 and f_4 functions generate the CK and IK, respectively, which are legacies of the 3G UMTS network. They are the keys to protect the confidentiality and integrity. However, in 4G LTE, there is only one session key K_{ASME}. Other keys are derived from this master session key. To reuse the infrastructure, K_{ASME} is derived from CK and IK by a key derivation function. Because the SQN may leak some information of the subscriber, the SQN is masked by an anonymity key generated by the function f_5. Notice that the SQNs may be mismatched, i.e. the received one is smaller than the stored value. When such situation happens, the UE and the AuC run a re-synchronization routine. For security reasons, the MAC and AK in re-synchronization are generated by f_1^* and f_5^*, respectively.

After the AKA and security mode set-up procedure, the integrity and confidentiality of all the control messages are protected [8]. The integrity protection method, called f_9, is applied in radio resource control (RRC) layer. The f_9 function takes four inputs: $COUNT-I$, $MESSAGE$, $DIRECTION$ and $FRESH$. $COUNT-I$ is a counter, which counts how many times the integrity key is used. After $COUNT - I$ reaches a threshold, the UE and MME must negotiate a new integrity key. $MESSAGE$, as the name suggests, is the message the MAC is computed on. $DIRECTION$ indicates whether the message is sent in the uplink or downlink. This mechanism can prevent replay attacks. $FRESH$ is the random number generated by the MME and sent to the UE during the (RRC) security mode command at the connection set-up procedure. This random number also helps the system avoiding replay attacks.

Upon the UE or other entities on the radio network receiving a message, it first checks the sequence number to see whether the message is fresh. If it is not, this message is discarded. Otherwise, UE or the network validates the message by checking MAC. If the message does not pass the validation, it is discarded.

1.3 Message Authentication Code

When the active wiretapper was proposed [112], integrity protection attracted more attention than ever before. Simmons [99] proposed the first authentication model using Message Authentication Code (MAC), and demonstrated two different attacking models against MAC: the impersonation forgery attack and the substitution forgery attack. Impersonation forgery means that the opponent can forge a MAC without intercepting any message-tag pairs, and substitution forgery means that the opponent can forge by observing message-tag pairs. The success rate of these two attacks are denoted by P_I and P_S, respectively.

There are several ways to construct a MAC, such as algebra [37, 82], combinatorics [101, 102], finite geometries [108], and coding theory [69].

Recently, people have begun to construct MACs on top of hash functions in one of two ways: the first is based on the universal hash functions, such as in Stinson's work [103]; the second uses cryptographic hash functions to construct MACs as Krawczyk's work [74]. Note that constructions using block ciphers also belong to the latter category, because block ciphers can be equivalent to keyed hash functions in the sense of CBC or other modes. For example, CBC-MAC [20] and XOR-MAC [19] are two MACs using block ciphers.

Originally, the keyed hash functions and block ciphers based MACs attracted most researchers' attention. As stream ciphers were widely deployed in the wireless communication related to our daily life, MACs based on stream ciphers became a hot reach topic. Constructions based on universal hash functions are naturally suitable for working with stream ciphers, because most of these constructions' security proofs require a one-time pad. Usually, the period of the key stream generated by a stream cipher is guaranteed to be huge. Therefore, the key stream can be considered as a one-time pad required by the universal-hash-function-based constructions.

Krawczyk demonstrated that any Almost XOR Universal (AXU) hash function is equivalent to a secure message authentication code [73]. Krawczyk's theory forms basis of many well-known MACs, such as GCM [82], EIA1 (UIA2) [3], etc. GCM has been standardized by NIST as an authenticated encryption (AE) scheme. EIA1 is the integrity protection algorithm deployed in the 4G LTE system. Note that an AE scheme encrypts and authenticates messages in one step. If we remove the encryption of GCM, it becomes GMAC, which is also a kind of MAC. EIA1 and GMAC are similar, because both are based on the evaluation of a polynomial. We call such MACs polynomial based MACs in this paper. The difference between these two is that EIA1 uses a method called secure truncation [21], while GCM just simply truncates the output to a fixed size.

In this section, we introduce two different kinds of MACs. Sub-section 1.3.1 presents the MACs based on block ciphers and hash functions, then Sub-section 1.3.2 demonstrates the MACs based on stream ciphers.

1.3.1 Message Authentication Codes Based on Block Ciphers and Hash Functions

Here we take CBC-MAC [28] and HMAC [15, 74] as examples to show the MACs based on block ciphers. Of course, there are other outstanding MACs based on block ciphers, but these are omitted from this thesis.

The structure of CBC-MAC is illustrated in Figure 1.3.

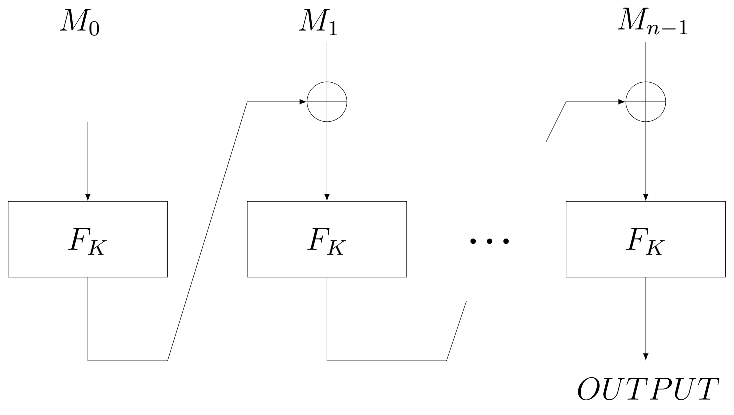


Figure 1.3: CBC-MAC

CBC-MAC is a variant of the Cipher Block Chain (CBC) mode of block cipher. The underlying cipher F_K can be any block cipher or keyed hash function that maps a fixed-length input to a fixed-length output. Because CBC-MAC is computed in a chain, the computation cannot be parallelized.

When CBC mode is used for both authentication and encryption, using the same key for both authentication and encryption is forbidden [54] because of the following attack. Assume that Alice encrypts the message

$$P = \{P_0, \cdots, P_{n-1}\}$$

to get the ciphertext

$$C = \{C_0, \cdots, C_{n-1}\},$$

and uses the last block C_{n-1} as the MAC. Now Alice sends C to Bob. However, Eve intercepts and modifies C to

$$C' = \{C'_0, \cdots, C'_{n-2}, C_{n-1}\}.$$

Note that only the last block remains unchanged. Bob receives C' and decrypts. What Bob gets is different from the original P, but he is unaware of this modification.

Even using different keys for encryption and authentication, CBC-MAC is vulnerable to another well-known attack [28]. Assume that the MAC is computed on the ciphertext

$$C = \{C_0, \cdots, C_{n-1}\},$$

which is encrypted by a different key. The tag is computed in a recursive way.

$$T_{n-1} = F_K(C_{n-1} \oplus T_{n-2}).$$

Eve can construct another message

$$C' = \{C_0, \cdots, C_{n-1}, T_{n-1} \oplus C_0, \cdots, C_{n-1}\}.$$

The tag computed on the first n blocks is T_{n-1}. When computing the n-th block, we have

$$T_n = F_K(T_{n-1} \oplus T_{n-1} \oplus C_0) = F_K(C_0),$$

where F_k is the same block cipher as in Figure 1.3. Then this message and the original message have the same MAC (assume the initial vector of CBC-MAC is zero).

Since so many attacks are possible, numerous variants of CBC-MAC have emerged, such as EMAC [52], OMAC[105], TMAC [75], etc. Most of these variants improve CBC-MAC by encrypting the last block.

Unlike CBC-MAC, HMAC requires its underlying primitive to have the capability of mapping arbitrary-length input to fixed-length output. Most cryptographic hash functions fulfill this requirement, but all block ciphers do not have such capability. Thus, block ciphers cannot be the underlying primitive of HMAC without any modification. However, if we modify block ciphers a little bit, such as using Merkle-Damgård construction [63], they can be utilized as the underlying cipher of HMAC. HMAC is defined by

$$T = H((K \oplus opad)||H((K \oplus ipad)||\overline{\mathbf{M}})),$$

where H is a cryptographic hash function or a block cipher, *opad* and *ipad* are two constants, K is a key.

The two-layer design is to avoid the length extension attack [15], which is a kind of attack applied to Merkle-Damgård hash constructions [42, 85, 86]. Merkle-Damgård hash constructions assembly the compression functions like a chain. If HMAC has only one layer, and the underlying cipher of HMAC is a Merkle-Damgård hash function, the length extension attack can also be applied to HMAC.

Because of the Merkle-Damgård construction, $K \oplus opad$ and $K \oplus ipad$ are prefixed to a message and its intermediate tag to increase the computation in an exhaustive search. For example, if HMAC is designed as

$$T = H(H(\overline{\mathbf{M}}||(K \oplus ipad))||(K \oplus opad)),$$

the adversary can pre-compute the hash value $t = H(\overline{\mathbf{M}})$, and the hash value of t. Since the hash values of $\overline{\mathbf{M}}$ and t are used in the computation of MAC, and these two values remain unchanged in the following computation, the pre-computation can reduce the time consumption of the exhaustive search.

1.3.2 Message Authentication Codes Based on Stream Ciphers

Several outstanding MACs, such as the one based on Grain-128 [10], cryptographic check sum [76], are based on stream ciphers. In this subsection, we introduce a very special case, GCM. This algorithm is proposed for the block cipher in counter mode. However, since the counter mode is actually a kind of stream ciphers, GCM can work not only with block ciphers, but also with arbitrary stream ciphers. Therefore, we consider it as a MAC based on stream ciphers. We remark here that GCM is an AE scheme. Therefore, the encryption and the authentication are done together. GCM has two blocks, GCTR and GHASH. GCTR encrypts messages, and GHASH generates the hash tag of messages. If the hash tag generated by GHASH is eventually encrypted by GCTR, GHASH and GCTR together are called GMAC.

Figure 1.4 demonstrates the structure of GCM. This figure takes AES as an example, but one can use any block cipher to replace AES in this figure. H is the hash sub key, which is generated by encrypting $\bar{\mathbf{0}}$ using the key K. The notations "$\boxplus$", "$\boxtimes$" and "$\oplus$" represent integer addition, multiplication over finite field and XOR respectively. J_0, the pre-counter block, is initialized by IV in a complicated way as follows. Let $s = 128\lceil len(IV)/128\rceil - len(IV)$. Then

$$J_0 = \begin{cases} IV||0^{31}||1, \text{ If } len(IV) = 96, \\ GHASH_H(IV||0^{s+64}||[len(IV)]_{64}), \text{ If } len(IV) \neq 96. \end{cases}$$

Several works analyzed GCM, such as Käsper and Schwabe [70], Iwata *et al.* [68], Zhu *et al.* [113], Abdelraheem *et al.*[9] and Saarinen [97].

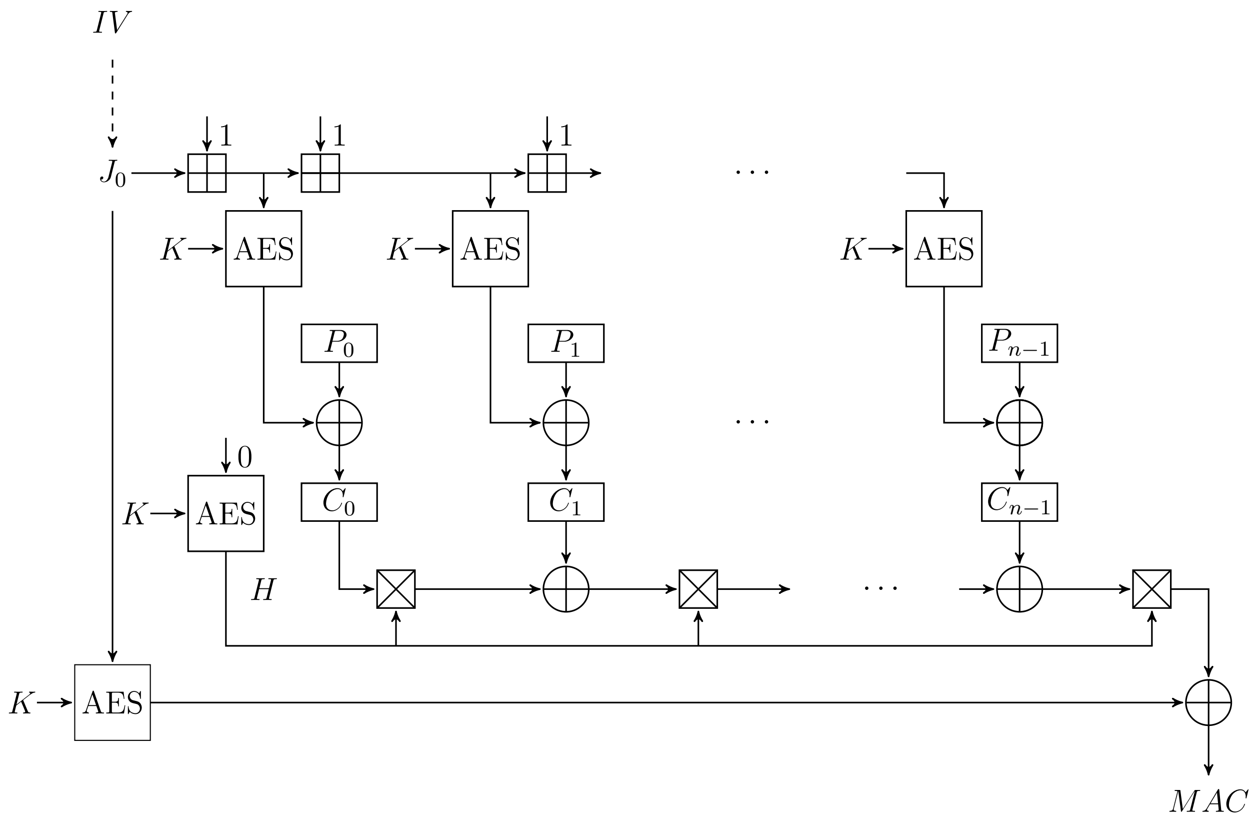


Figure 1.4: GCM

1.4 Related Work

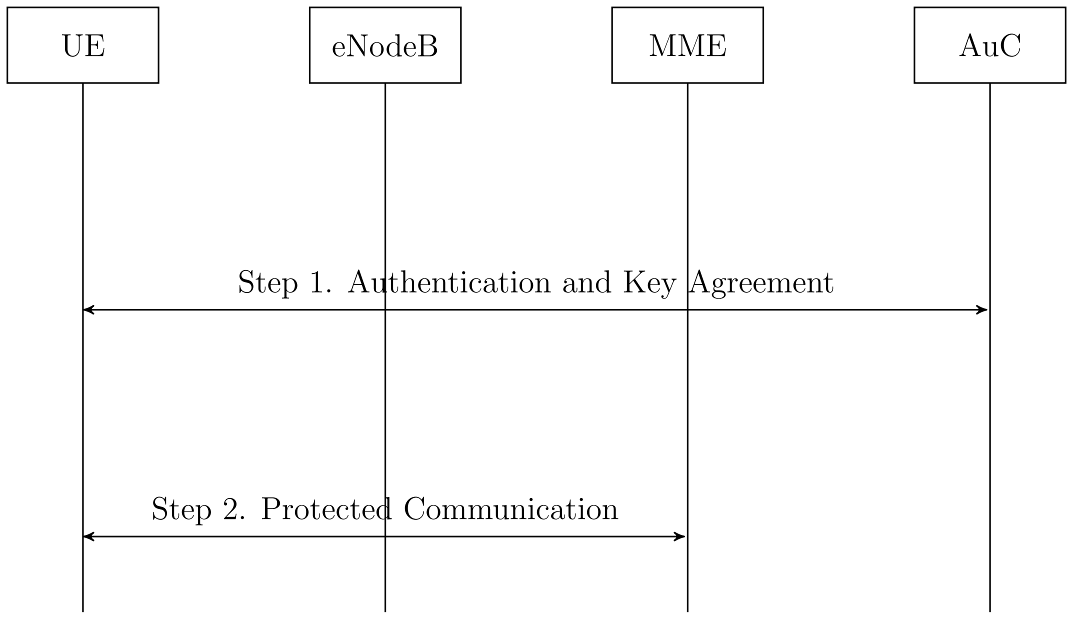

UE
eNodeB
MME
AuC
Step 1. Authentication and Key Agreement
Step 2. Protected Communication

Figure 1.5: The Security Architecture of 4G LTE

As shown in Figure 1.5, this thesis is about the integrity and authenticity protection in both Steps 1 and 2. The first step authenticates entities on the network and derives the session key, which will be used in Step 2. Both entity authentication and message authentication methods called A3/A8 are applied to Step 1. Step 2 is the communication after the security set-up in Step 1. The message integrity of Step 2 is protected by MAC using the key derived in Step 1.

Security Analysis on Keccak and TUAK

For Step 1, Vodafone has proposed an algorithm set called TUAK [53] to do the authentication and message authentication, recently. TUAK is built upon Keccak [25], which was standardized as SHA-3 by NIST. The design of the Keccak hash function is based on the sponge function and its internal state size is 1600 bits [23, 91]. The Keccak cryptographic hash algorithm set contains four instances: Keccak-224, Keccak-256, Keccak-384 and Keccak-512 [91]. Several cryptanalytic attacks including differential attacks and distinguishing attacks have been developed on the round-reduced Keccak hash function [12, 22, 44–46, 49, 77, 87, 89, 104]. A differential cryptanalytic technique is a method that can be used to produce a collision of a hash function. The differential attack for building a distinguisher or producing collisions on round-reduced Keccak can be found in several papers [44, 45, 49, 89]. Dinur *et al.* [44] proposed a collision attack on 4-round Keccak-224 and Keccak-256, and a near-collision attack on 5-round Keccak-224 and Keccak-256. In another paper of Dinur *et al.* [45], the authors proposed a practical collision attack on 3-round Keccak-384 and Keccak-512, a collision attack on 4-round Keccak-384 with complexity of 2^{147} and a collision attack on 5-round Keccak-256 with complexity of 2^{115}. Daemen and Van Assche studied the differential trails of the Keccak hash function in their paper [41]. Recently, Dinur *et al.* applied the cube attack on a message authentication code based on 6-round Keccak [46]. TUAK is a new application of Keccak. Although Keccak is proved to be secure, it does not necessarily mean TUAK is secure as well. This thesis proposes security proof and security assessment of TUAK for the first time to prove TUAK is as secure as Keccak.

Security Analysis of EIA1 and EIA3

In Step 2, message integrity is protected by the EIA family [3, 6]. EIA1 and EIA3 are two MACs based on stream ciphers. EIA1 is a kind of polynomial evaluation MAC, which computes the tag by evaluating a polynomial. Saarinen *et al.* presented an attack called cycling attack, and demonstrated the application of this attack on polynomial evaluation MACs in their paper [97]. Handschuh *et al.* showed key-recovery attacks [62] on universal hash function based MACs. The authors used two different methods: weak key and birthday attacks. The cycling attack is a special case of the weak key attacks proposed by Handschuh *et al.* However, no attacks can be directly applied to EIA1, and cause serious security flaws. EIA3 is equivalent to a polynomial evaluation MAC. The argument of EIA1 applies to EIA3 as well. Besides, a forgery attack [106] proposed by Thomas *et al.* could forge MACs generated by EIA3 V1.3. The current version of EIA3 is V1.5, which is resistant against such attacks. In a nutshell, none of the existing attacks harm the security of EIA1 and EIA3 seriously. The forgery attack called linear forgery attack proposed in this thesis is the first applicable attack on EIA1 and EIA3.

MAC Design

MAC is used in every stage of the communication in Figure 1.5. Thereby, secure algorithms are highly demanded. Since the linear forgery attack is a relatively new attack, which has not attracted much attention, two MACs proposed in this thesis are the first work resistant to such attacks. The first MAC is a variant of EIA1, and replaces the addition over a finite filed with the addition over a ring. There does not exist any work discussing such kind of MACs. The second MAC is based on APN functions, whose definition will be presented in Chapter 2. Chanson *et al.* [37] proposed MACs based on functions with optimal nonlinearity for the first time. Ding *et al.* [43] and Carlet *et al.* [34] continued the work of Chanson *et al.* Jian *et al.* [79] showed that the relationship between the perfect nonlinearity functions and the universal hash functions, and constructed an authentication code based on the universal hash function. None of the above MAC constructions can take an arbitrary-length input. The second MAC proposed in this thesis is the first MAC based on APN functions, and maps arbitrary-length inputs to fixed-length outputs just like CBC-MAC, HMAC, etc.

1.5 Contributions

Contributions of this thesis are listed as follows.

1. *Linear Forgery Attack:* This is the first practical attack on EIA1 and EIA3. Known two message-MAC pairs, attackers can forge up to 2^{32} valid message-MAC pairs. This thesis also demonstrates an example that attackers can easily find meaningful messages among those 2^{32} valid message-MAC pairs. By a well-designed scenario, attackers can apply the linear forgery attack to the real system. This practical attack makes the linear forgery attack more serious than existing attacks.

2. *Security Proof of Almost XOR Universal MAC and EIA1:* The security proof of Almost XOR Universal (AXU) MAC left a gap between the AXU property and the probability of substitution forgery attacks. We fix this flaw, and show AXU MAC is secure under certain attack model. EIA1 as an AXU MAC is secure under this attack model. In addition, we consider the security of EIA1 under other attack models. This is the first time that EIA1 is examined under different models.

3. *Improvement of EIA1:* We point out several implementation flaws of EIA1 official implementation, and improve the official implementation by optimizing the multiplication over the finite field. Moreover, we introduce an efficient polynomial evaluation algorithm into EIA1 to make it faster than the official implementation using the Horner's Rule.

4. *Security Proof of TUAK:* TUAK is a new algorithm set. The security proof presented in this thesis is the first study on its security. This thesis considers the MAC algorithm and key derivation functions (KDFs) in TUAK, and proves the security of each algorithm. Based on the Keccak's main document, this thesis suggests a tiny modification to TUAK to make it more secure.

5. *Multi-Output Filtering Model and Its Applications:* The Multi-Output Filtering Model (MOFM) is originally used in the communication system. We novelly apply it to the analysis of cryptographic primitives for the first time. This model utilizes LFSR as input, and arbitrary cryptographic primitive as the filtering function. By experiment, the MOFM may reveal some randomness of the primitive. For example, we test the linear complexities and algebra degrees of four primitives in this thesis. To test the linear complexity, we take the output as sequences, and compute the linear complexity for each component sequence. The distribution of the linear complexity shows differences between different primitives. For the algebra degree, we take the four primitives as vector Boolean functions, and compute the degree of each component function.

6. *MAC Designs:* The contribution is twofold. The first called WGIA-128 is a variant of EIA1. WGIA-128 replaces the addition over finite filed used in EIA1 with the addition over ring, which makes WGIA-128 resist the linear forgery attack. The second is AMAC, which is MAC based on APN functions. There exist many MAC constructions based on APN functions. However, all the existing works can deal the fixed-length message only. AMAC is the first $\mathbb{F}_2^*$-to-$\mathbb{F}_2^m$ MAC, like CBC-MAC or HMAC, based on APN functions. There are two different constructions of AMAC, and both have security proofs. The efficiency of AMAC is better than EIA1.

1.6 Organization

This thesis is organized as follows. Chapter 2 introduces the preliminaries, including the notations, definitions, etc. Chapter 3 presents a forgery attack on EIA1. This attack exploits the linear structure of EIA1. Inspired by the attack in Chapter 3, Chapter 4 analyzes the security of EIA1 under different security models. Moreover, the implementation of EIA1 is improved by importing an efficient polynomial evaluation algorithm, which is better than Horner's Rule. The next two chapters (5 and 6) discuss another set of authentication and message authentication methods used in the 4G LTE system. The algorithm set, very recently proposed by Vodafone, is called TUAK. Chapter 5 is the security proof of TUAK as MAC and KDF. Chapter 6 introduces a new analysis method called Multi-Output Filtering Model (MOFM), which is applied to TUAK later in this chapter. Chapter 7 presents two new MACs. One, an variant of EIA1, is resistant to the attack we propose in Chapter 3. The other MAC is based on APN functions. The last chapter, Chapter 8, concludes this thesis and points out some open problems, which are still unsolved.

Chapter 2

Preliminaries

This chapter presents the preliminaries of this thesis. Section 2.1 lists all notations used in the chapters that follow. Section 2.2 introduces three cryptographic primitives: Snow3G, ZUC, and WG-16. Snow3G and ZUC are both stream ciphers deployed in the 4G LTE system. They are the underlying ciphers of EIA1 and EIA3, two MACs discussed in this thesis. WG-16 is another stream cipher. Because one of our MAC designs is based on WG-16 (using its uniform distribution property), we demonstrate it in this chapter. Section 2.4 presents the concept of universal hash families, including universal and almost universal hash, strongly universal and almost strongly universal hash, and almost xor universal hash.

2.1 Notations

Table 2.1 lists the notations used in this thesis.

2.2 Underlying Ciphers

In this section, we present three stream ciphers: Snow3G, ZUC, and WG-16. The first two are the underlying ciphers of EIA1 and EIA3, respectively. We introduce them here because their structures can help us to understand the performance of EIA1 and EIA3 in Chapter 7. WG-16 is another stream cipher, and is the cipher underlying the MAC proposed in this research. To understand the security proof of this MAC, we demonstrate WG-16 and its properties in this section.

Notations	Explanation
$GF(p^n)$ or $\mathbb{F}_{p^n}$	The Galois Field with p^n elements, where p is a prime.
$\mathbb{F}_2^n$	The n-dimensional vector space over $\mathbb{F}_2$.
2^{32}	2^{32}
$Tr_m^n(x)$	The trace function, which maps element from $\mathbb{F}_{q^n}$ to $\mathbb{F}_{q^m}$.
$Tr(x)$	The absolute trace, which maps element to the ground field.
$\overline{\mathbf{M}} = \{M_0, \cdots, M_{l-1}\}$, where $M_i \in \mathbb{F}_{2^n}$, for $0 \leq i < l$	$\overline{\mathbf{M}}$ is a vector over $\mathbb{F}_{2^n}$.
$\oplus$	The XOR operator.
a + b $a, b \in \mathbb{F}_{2^n}$	The addition over a Galois Field.
a + b $a, b \in \mathbb{Z}$	The integer addition.
$a \cdot b$ or ab $a, b \in \mathbb{F}_{2^n}$	The multiplication over a Galois Field. Without ambiguity, the "$\cdot$" can be removed.
$a * b$ or $a \times b$	The integer multiplication.
$a \| b$	Concatenation of a and b.
$a \overset{\$}{\leftarrow} \mathcal{A}$, where $\mathcal{A}$ is a set.	a is randomly selected from $\mathcal{A}$.

Table 2.1: Notations

2.2.1 Snow3G and ZUC

Snow3G

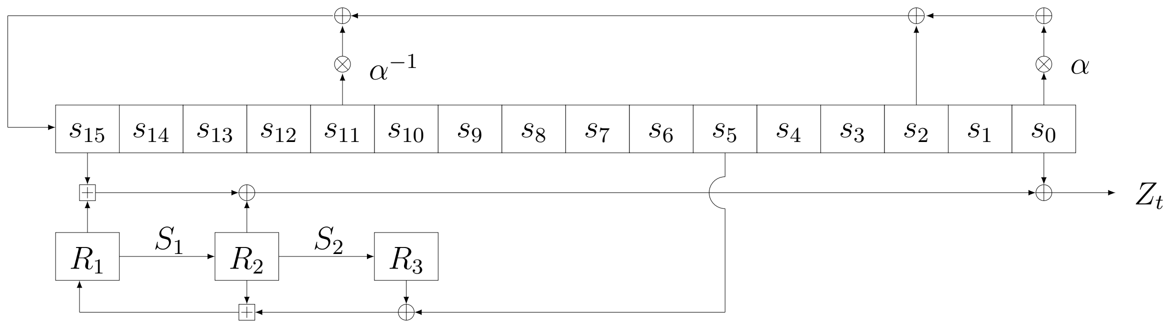
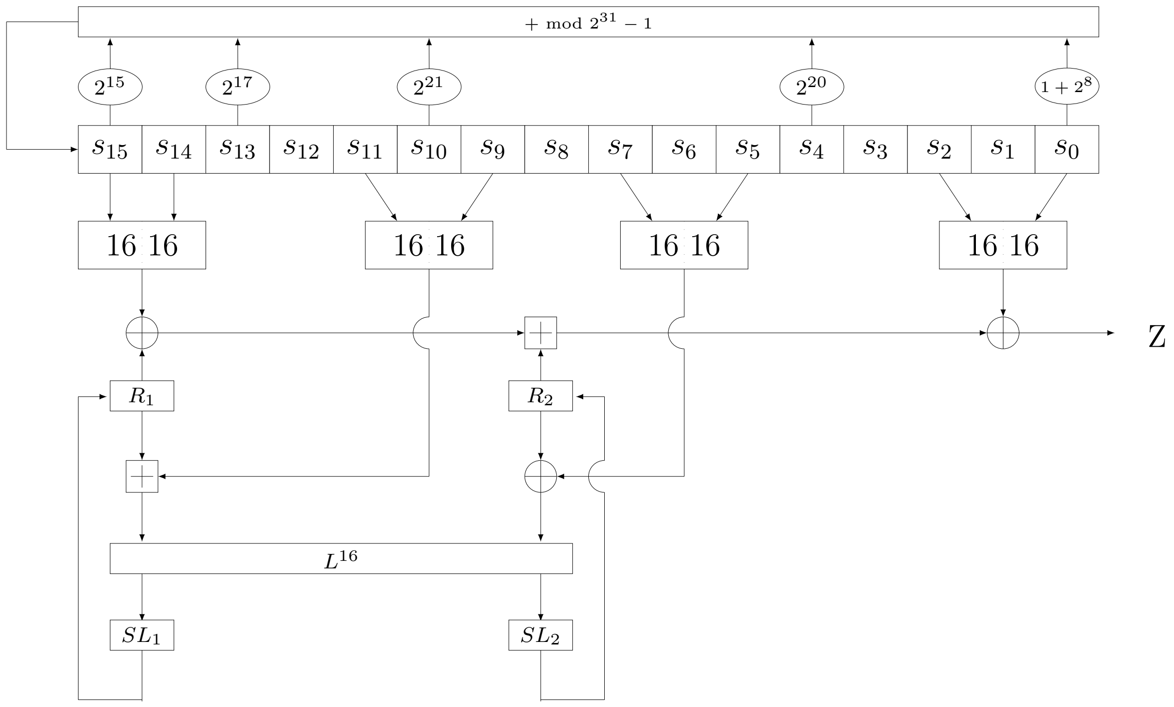

α^{-1}
α
s_{15} s_{14} s_{13} s_{12} s_{11} s_{10} s_9 s_8 s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0
Z_t
R_1
S_1
R_2
S_2
R_3

Figure 2.1: Snow3G

Snow3G [2] is composed of a 16-stage linear feedback shift register (LFSR) and a finite state machine (FSM). Since each register of the LFSR can hold 32 bits, the operation of the LFSR is defined over $GF(2^{32})$. To define $GF(2^{32})$, we first define $GF(2^8)$ by the polynomial $g_1(x) = x^8+x^7+x^5+x^3+1$ over $GF(2)$. Then we extend $GF(2^8)$ to $GF(2^{32})$ by the polynomial $g_2(x) = x^4+\beta^{23}x^3+\beta^{245}x^2+\beta^{48}x+\beta^{239}$ over $GF(2^8)$, where $\beta \in GF(2^8)$ is the root of $g_1(x)$. The feedback polynomial of this LFSR is $f(x) = \alpha x^{16}+x^{14}+\alpha^{-1}x^5+1$, where $\alpha \in GF(2^{32})$ is the root of $g_2(x)$. The FSM is composed of three 32-bit registers and two S-boxes. These two S-boxes, called S_1 and S_2, map 32 bits to 32 bits. S_1 and S_2 are made of four juxtaposed 8-bit S-boxes, which rely on the Rijndael S-box, Dickson polynomial, and MixColumn operation. Notice that Snow3G combines the addition in $GF(2^{32})$ together with the integer modular addition. To balance the key stream, the output of FSM is masked by the register s_0. From the description above, we can see that Snow3G is a kind of nonlinear filter generator. The key stream is generated by passing the internal states of the LFSR through a nonlinear filtering function.

In the evaluation report on Snow3G [1], the authors proved only the period of the key stream generated by Snow3G, but no other randomness properties, such as k-tuple distribution, run distribution, etc. They also showed some properties of S-boxes and claimed that Snow3G resists distinguishing attacks and algebraic attacks. At the end of the evaluation report, the authors compared Snow3G with Kasumi and indicated that Snow3G has better throughput than Kasumi.

ZUC



+ mod $2^{31}-1$
2^{15}
2^{17}
2^{21}
2^{20}
$1+2^8$
s_{15} s_{14} s_{13} s_{12} s_{11} s_{10} s_9 s_8 s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0
16 16
16 16
16 16
16 16
Z
R_1
R_2
L^{16}
SL_1
SL_2

Figure 2.2: ZUC

Similar to Snow3G, ZUC [7] is a nonlinear filtering generator as well. It also has a 16-stage LFSR and a FSM. However, the operation of the LFSR is defined over a prime field $GF(2^{31}-1)$ (Snow3G is defined over $GF(2^{32})$). Thus, each register has only 31 bits. The feedback polynomial is $f(x) = 2^{15}x^{15} + 2^{17}x^{13} + 2^{21}x^{10} + 2^{20}x^4 + 2^8 + 1$. Because the registers of the LFSR are 31 bits each, which is not a power of 2, a bit reorganization layer is placed between the LFSR and the FSM in order to make each word 32 bits. The FSM has two 32-bit registers, which hold the intermediate states of the FSM. The nonlinear layer of the FSM is made up of the Rijndael S-box, Dickson polynomial, MixColumn, and ShiftRow.

2.2.2 WG-16

WG-16 is an efficient variant of the WG stream cipher family with 128-bit secret key and 128-bit initial vector (IV). It consists of a 32-stage LFSR with the feedback polynomial $l(x)$ followed by a WG-16 transformation module with decimation $d = 1057$. Therefore, it can be regarded as a nonlinear filter generator over finite field $\mathbb{F}_{2^{16}}$. WG-16 operates in two phases, namely an initialization phase and a running phase.

Initialization Phase

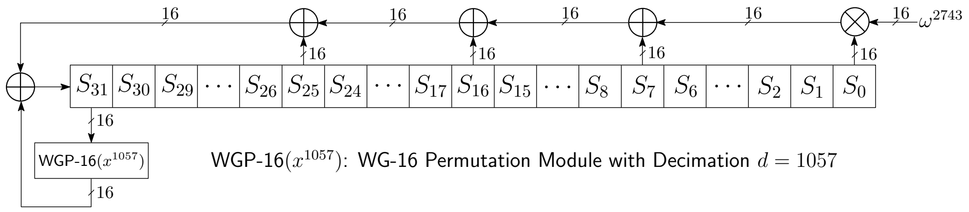

16
16
16
16
16
ω^{2743}
S_{31} S_{30} S_{29} $\cdots$ S_{26} S_{25} S_{24} $\cdots$ S_{17} S_{16} S_{15} $\cdots$ S_8 S_7 S_6 $\cdots$ S_2 S_1 S_0
WGP-16(x^{1057})
WGP-16(x^{1057}): WG-16 Permutation Module with Decimation $d = 1057$

Figure 2.3: The Initialization Phase of the Stream Cipher WG-16

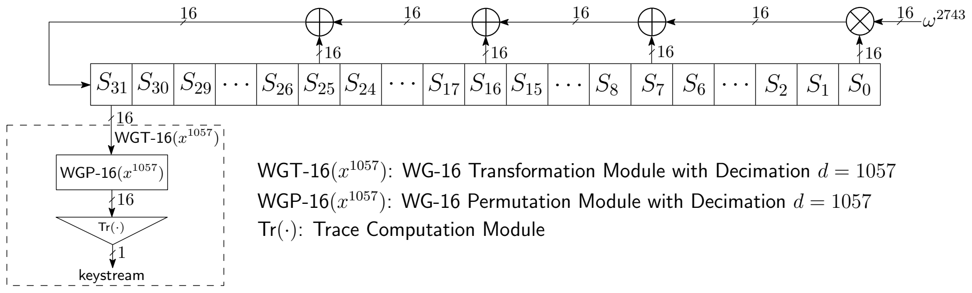

16
16
16
16
16
ω^{2743}
S_{31} S_{30} S_{29} $\cdots$ S_{26} S_{25} S_{24} $\cdots$ S_{17} S_{16} S_{15} $\cdots$ S_8 S_7 S_6 $\cdots$ S_2 S_1 S_0
WGT-16(x^{1057})
WGP-16(x^{1057})
Tr(·)
1
keystream
WGT-16(x^{1057}): WG-16 Transformation Module with Decimation $d = 1057$
WGP-16(x^{1057}): WG-16 Permutation Module with Decimation $d = 1057$
Tr(·): Trace Computation Module

Figure 2.4: The Running Phase of the Stream Cipher WG-16

The key/IV initialization phase of WG-16 is illustrated in Fig. 2.3. Let the 128-bit secret key be $K = (K_{127}, \ldots, K_0)_2$, the 128-bit IV be $IV = (IV_{127}, \ldots, IV_0)_2$, and the internal state of the LFSR be $S_0, \ldots, S_{31} \in \mathbb{F}_{2^{16}}$, where $S_i = (S_{i,15}, \ldots, S_{i,0})_2$ for $i = 0, \ldots, 31$. The key/IV initialization process is conducted below:

$$S_i = \begin{cases} (K_{8i+7}, \ldots, K_{8i}, IV_{8i+7}, \ldots, IV_{8i})_2 & i = 0, \ldots, 15, \\ S_{i-16} & i = 16, \ldots, 31. \end{cases}$$

Once the LFSR is loaded with the key/IV, the apparatus runs for 64 clock cycles. During each clock cycle, the 16-bit internal state S_{31} is sent to the WG-16 permutation with decimation $d = 1057$ (i.e., the $\mathsf{WGP\text{-}16}(x^{1057})$ module) and the output is used as the nonlinear feedback to update the internal state of the LFSR. The LFSR state update procedure follows the recursive relation:

$$\begin{aligned} S_{k+32} = \big(\omega^{2743} \otimes S_k\big) \oplus S_{k+7} \oplus S_{k+16} \oplus \\ S_{k+25} \oplus \mathsf{WGP\text{-}16}\left(S_{k+31}^{1057}\right), \qquad 0 \leq k < 64. \end{aligned}$$

After the initialization phase, WG-16 starts the running phase and a bit key stream is generated per clock cycle.

Running Phase

The running phase of WG-16 is shown in Fig. 2.4. During the running phase, the 16-bit internal state S_{31} is sent to the WG-16 transformation with decimation $d = 1057$ (i.e., the $\mathsf{WGT\text{-}16}(x^{1057})$ module) and the output is a bit key stream. Note that the only feedback in the running phase is within the LFSR and the recursive relation for updating the internal state of LFSR is given below:

$$S_{k+32} = \big(\omega^{2743} \otimes S_k\big) \oplus S_{k+7} \oplus S_{k+16} \oplus S_{k+25}, \quad k \geq 64.$$

The $\mathsf{WGT\text{-}16}(x^{1057})$ module comprises of two sub-modules: a WG-16 permutation module $\mathsf{WGP\text{-}16}(x^{1057})$ followed by a trace computation module $\mathrm{Tr}(\cdot)$. While the $\mathsf{WGP\text{-}16}(x^{1057})$ module permutes elements over $\mathbb{F}_{2^{16}}$, the $\mathrm{Tr}(\cdot)$ module compresses a 16-bit input to a bit key stream.

Randomness Properties of the WG-16 Key stream

The key stream generated by the stream cipher WG-16 has the following desired randomness properties [38]:

1. The key stream has a period of $2^{512} - 1$.
2. The key stream is balanced, i.e., the number of 0's is only one less than the number of 1's in one period of the key stream.
3. The key stream is an ideal two-level autocorrelation sequence.
4. The key stream has an ideal t-tuple ($1 \leq t \leq 32$) distribution, i.e., every possible output t-tuple is equally likely to occur in one period of the key stream.
5. The linear span of the key stream can be determined exactly, which is $2^{79.046}$.

2.3 Details of TUAK, EIA1 and EIA3

2.3.1 TUAK

TUAK is an algorithm set for the 3GPP authentication and key derivation functions. TUAK specification contains the functions TOP_c, f_1, f_1^*, f_2, f_3, f_4, f_5 and f_5^* and all of them are built upon the Keccak-$f[1600]$ permutation. For the convenience, we first list the following notations, which will be used throughout this thesis. They are the same as those notations in the specification of TUAK.

- The permutation Keccak-$f[1600]$ is denoted by Π. Throughout this thesis we use Π to denote the Keccak permutation. According to the design of Π, it accepts an input of size 1600 bits that is represented by $\text{IN}[0], \cdots, \text{IN}[1599]$ and outputs an element of 1600 bits that is represented by $\text{OUT}[0], \cdots, \text{OUT}[1599]$;

- TOP is a 128-bit value decided by the operator and used as the input of the TOP_c function;

- ALGONAME is a fixed binary string of 56 bits, whose value can be found in the specification;

- INSTANCE is a binary variable of 8 bits. It uses to instantiate different functions, TOP_C, f_i's, f_1^* and f_5^*, in the TUAK algorithm set;

- K denotes the subscriber key;

In the following, we provide a description of each algorithm in TUAK. In this thesis, we interchangeably use the terms "algorithm" and "function" for the functions in the TUAK algorithm set.

Description of TOP_C

Authentication and key derivation functions of TUAK use the output of TOP_C. We provide a description of the TOP_C function. The TOP_c function takes a 256-bit value called TOP (chosen by the operator) and the subscriber key (can be 128 or 256 bits) as inputs (the other bits are constants), and outputs a 256 bit value TOP_c. More precisely, the inputs of TOP_c are assigned as follows:

- The value of INSTANCE is given by

$$\begin{aligned}\text{INSTANCE}[0]..\text{INSTANCE}[6] &= 0,0,0,0,0,0,0;\\ \text{INSTANCE}[7] &= 0 \text{ if the length of } K \text{ is } 128,\\ &= 1 \text{ if the length of } K \text{ is } 256.\end{aligned}$$

- IN[0]..IN[255] = TOP[255]..TOP[0];

- IN[256]..IN[263] = INSTANCE[7]..INSTANCE[0];

- IN[264]..IN[319] = ALGONAME[55]..ALGONAME[0];

- IN[i] = 0, for $320 \le i \le 511$;

- IN[512]..IN[767] = $K[255]..K[0]$ if the length of K is 256 bits;

- IN[512]..IN[639] = $K[127]..K[0]$ if the length of K is 128 bits ;

- IN[i] = 0 for $640 \le i \le 767$ if the length of K is 128 bits ;

- IN[i] = 1 for $768 \le i \le 772$;

- IN[i] = 0 for $773 \le i \le 1086$;

- IN[1087] = 1;

- IN[i] = 0 for $1088 \le i \le 1599$.

Figure 2.5 depicts an overview of an input assignment to the TOP$_C$ function. The TOP_C function is given by

$$\text{OUT} = \Pi(\text{INPUT})$$

with

$$TOP_c[0], .., TOP_c[255] = OUT[255], .., OUT[0].$$

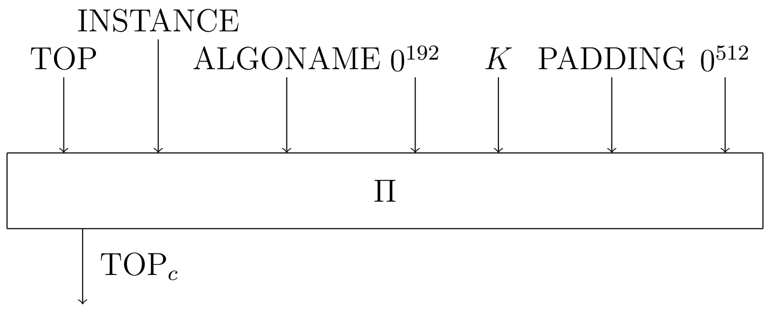


Figure 2.5: The TOP_c function of TUAK

Description of f_1

The function f_1 is used to generate the MACs. An input of 1600 bits to f_1 is constructed by the following binary strings: TOP_c (256 bits, generated by the function TOP_c), INSTANCE (8 bits), ALGONAME (56 bits), RAND (128 bits), AMF (16 bits), SQN (48 bits), and the subscriber key K (128 or 256 bits). The output value MAC can be 64, 128 and 256 bits. Precisely:

- The value of INSTANCE is:

$$\begin{aligned}
\text{INSTANCE[0]}, \text{INSTANCE[1]} &= 0, 0\\
\text{INSTANCE[2]..INSTANCE[4]} &= 0, 0, 1 \text{ if the MAC length is 64 bits}\\
&= 0, 1, 0 \text{ if the MAC length is 128 bits}\\
&= 1, 0, 0 \text{ if the MAC length is 256 bits}\\
\text{INSTANCE[5]}, \text{INSTANCE[6]} &= 0, 0\\
\text{INSTANCE[7]} &= 0 \text{ if the length of } K \text{ is 64 or 128}\\
&= 1 \text{ if the length of } K \text{ is 256.}
\end{aligned}$$

- IN[0]..IN[255] = TOP_c[255]..TOP_c[0];

- IN[256]..IN[263] = INSTANCE[7]..INSTANCE[0];

- IN[264]..IN[319] = ALGONAME[55]..ALGONAME[0];

- IN[320]..IN[447] = RAND[127]..RAND[0];

- IN[448]..IN[463] = AMF[15]..AMF[0];
- IN[464]..IN[511] = SQN[47]..SQN[0];
- IN[512]..IN[767] = K[255]..K[0] if the length of K is 256 bits ;
- IN[512]..IN[639] = K[127]..K[0] if the length of K is 128 bits ;
- IN[i] = 0 for $640 \leq i \leq 767$ if the length of K is 128 bits ;
- IN[i] = 1 for $768 \leq i \leq 772$;
- IN[i] = 0 for $773 \leq i \leq 1086$;
- IN[1087] = 1;
- IN[i] = 0 for $1088 \leq i \leq 1599$.

A high-level overview of the input assignment is provided in Figure 2.6. The MAC function f_1 is defined as

$$\mathrm{OUT} = \Pi(\mathrm{IN}).$$

The output of f_1, i.e. the MAC, can be of length 64, 128, and 256 and is given by

$$\begin{aligned}
\mathrm{MAC}[0]..\mathrm{MAC}[63] &= \mathrm{OUT}[63]..\mathrm{OUT}[0], \text{ if the MAC length is 64 bits,}\\
\mathrm{MAC}[0]..\mathrm{MAC}[127] &= \mathrm{OUT}[127]..\mathrm{OUT}[0], \text{ if the MAC length is 128 bits,}\\
\mathrm{MAC}[0]..\mathrm{MAC}[255] &= \mathrm{OUT}[255]..\mathrm{OUT}[0], \text{ if the MAC length is 256 bits.}
\end{aligned}$$

Description of f_2 to f_5

The f_2 function is used to generate a response (RES) over a random number, a sequence number (SQN), and an AMF for a fixed key. The f_3, f_4 and f_5 functions are used to generate a cipher key (CK), an integrity key (IK) and an anonymity key (AK), respectively for a random number. The input assignment of these functions are given below.

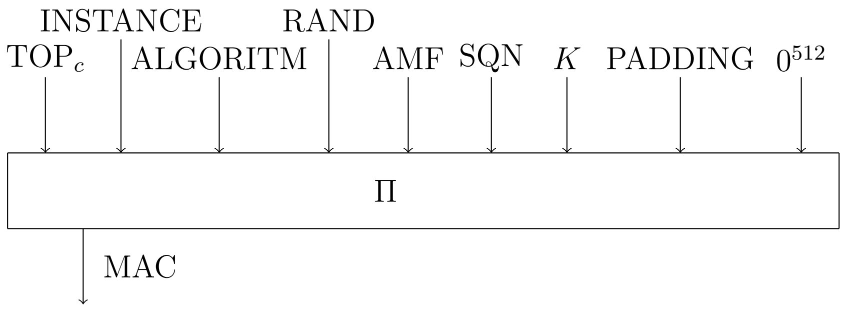


Figure 2.6: The f_1 function of TUAK for generating MAC

- The value of INSTANCE is:

$$\begin{aligned}
\text{INSTANCE}[0], \text{INSTANCE}[1] &= 0, 1 \\
\text{INSTANCE}[2]..\text{INSTANCE}[4] &= 0, 0, 0 \text{ if the RES length is 32 bits} \\
&= 0, 0, 1 \text{ if the RES length is 64 bits} \\
&= 0, 1, 0 \text{ if the RES length is 128 bits} \\
&= 1, 0, 0 \text{ if the RES length is 256 bits} \\
\text{INSTANCE}[5] &= 0 \text{ if the length of CK is 128 bits} \\
&= 1 \text{ if the length of CK is 256 bits} \\
\text{INSTANCE}[6] &= 0 \text{ if the length of IK is 128 bits} \\
&= 1 \text{ if the length of IK is 256 bits} \\
\text{INSTANCE}[7] &= 0 \text{ if the length of K is 128 bits} \\
&= 1 \text{ if the length of K is 256 bits.}
\end{aligned}$$

- IN[0]..IN[255] = TOP$_c$[255]..TOP$_c$[0];
- IN[256]..IN[263] = INSTANCE[7]..INSTANCE[0];
- IN[264]..IN[319] = ALGONAME[55]..ALGONAME[0];
- IN[320]..IN[447] = RAND[127]..RAND[0];
- IN[i] $= 0, 448 \leq i \leq 511$;
- IN[512]..IN[767] = K[255]..K[0] if the length of K is 256 bits ;

- IN[512]..IN[639] = $K[127]..K[0]$ if the length of K is 128 bits ;
- IN[i] = 0 for $640 \leq i \leq 767$ if the length of K is 128 bits ;
- IN[i] = 1 for $768 \leq i \leq 772$;
- IN[i] = 0 for $773 \leq i \leq 1086$;
- IN[1087] = 1;
- IN[i] = 0 for $1088 \leq i \leq 1599$.

On receiving the input INPUT, the outputs of $f_2 - f_5$ and f_5^* are calculated as follows

$$\text{OUT} = \Pi(\text{INPUT}).$$

The output of f_2 = RES, where:

$$\begin{gathered}
\text{RES}[0]..\text{RES}[31] = \text{OUT}[31]..\text{OUT}[0] \text{ if the RES length is 32 bits} \\
\text{RES}[0]..\text{RES}[63] = \text{OUT}[63]..\text{OUT}[0] \text{ if the RES length is 64 bits} \\
\text{RES}[0]..\text{RES}[127] = \text{OUT}[127]..\text{OUT}[0] \text{ if the RES length is 128 bits} \\
\text{RES}[0]..\text{RES}[255] = \text{OUT}[255]..\text{OUT}[0] \text{ if the RES length is 256 bits}
\end{gathered}$$

The output of f_3 = CK, where:

$$\begin{gathered}
\text{CK}[0]..\text{CK}[127] = \text{OUT}[383]..\text{OUT}[256] \text{ if the CK length is 128 bits} \\
\text{CK}[0]..\text{CK}[255] = \text{OUT}[511]..\text{OUT}[256] \text{ if the CK length is 256 bits}
\end{gathered}$$

The output of f_4 = IK, where:

$$\begin{gathered}
\text{IK}[0]..\text{IK}[127] = \text{OUT}[639]..\text{OUT}[512] \text{ if the IK length is 128 bits} \\
\text{IK}[0]..\text{IK}[255] = \text{OUT}[767]..\text{OUT}[512] \text{ if the IK length is 256 bits}
\end{gathered}$$

The output of $f5$ = AK, where:

$$\text{AK}[0]..\text{AK}[47] = \text{OUT}[815]..\text{OUT}[768]$$

Description of f_5^*

For the f_5^* function, the INSTANCE is given by

$$\begin{aligned} \text{INSTANCE}[0], \text{INSTANCE}[1] &= 1, 1 \\ \text{INSTANCE}[2], ..., \text{INSTANCE}[6] &= 0, 0, 0, 0, 0 \\ \text{INSTANCE}[7] &= 0 \text{ if the length of K is 128 bits} \\ &= 1 \text{ if the length of K is 256 bits.} \end{aligned}$$

The assignment of INPUT is the same as the input assignment of $f_2 - f_5$ with the above INSTANCE and the following changes

$$\text{IN}[257] = 0, \text{IN}[258] = 0, \text{IN}[259] = 0, \text{IN}[260] = 0, \text{IN}[261] = 0, \text{IN}[263] = 1.$$

The output of f_5^* is given by

$$\text{OUT} = \Pi(\text{INPUT})$$

where

$$\text{AK}[0]..\text{AK}[47] = \text{OUT}[815]..\text{OUT}[768].$$

Different algorithms of TUAK produce outputs of different lengths. We denote by f_i-M the f_i function/algorithm with output M bits.

2.3.2 EIA1

Figure 2.7 shows the structure of EIA1. $P, Q \in GF(2^{64})$ and $OTP \in GF(2^{32})$ are generated by Snow3G.

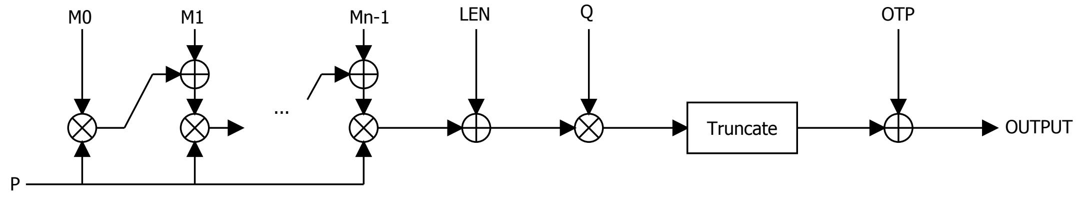


Figure 2.7: EIA1

According to Figure 2.7, the MAC of the message

$$\overline{\mathbf{M}} = \{M_0, \cdots, M_{n-1}\}, \text{ where } M_i \in GF(2^{64}),$$

is given by

$$\big[(\sum_{i=0}^{n-1} P^{n-i} M_i + LEN)Q\big]_{Truncate} + OTP, \tag{2.1}$$

where LEN is an element in $GF(2^{64})$. By the specification [3], "Truncate" means cutting the least significant 32 bits and leaving the most significant 32 bits. Specifically, if $H \in GF(2^{64})$ and $H = h_1 || h_0$, where h_0 and h_1 are the least significant and the most significant 32 bits respectively, $Truncate(H) = h_1$.

Lemma 1. *Let $p(x) = x^2 + ax + b$ be the defining polynomial of $GF(2^{64})$ over $GF(2^{32})$, where $a, b \in GF(2^{32})$. Then*

$$Truncate(H) = Tr_{32}^{64}(H)a^{-1},$$

where $Tr_{32}^{64}(x) = x^{2^{32}} + x$ is the trace function from $GF(2^{64})$ to $GF(2^{32})$.

Proof. Let α be a root of $p(x)$ over $GF(2^{64})$. Thus we have

$$\alpha^2 + a\alpha = b. \tag{2.2}$$

Let $H = h_1 + h_2\alpha$. Since $h_1, h_2 \in GF(2^{32})$,

$$Tr_{32}^{64}(H) = h_1 + h_2\alpha^{2^{32}} + h_1 + h_2\alpha = h_2(\alpha^{2^{32}} + \alpha).$$

Because of Eqn.(2.2),

$$\begin{aligned}
\left(\frac{\alpha}{a}\right)^2 + \frac{\alpha}{a} &= \frac{b}{a^2} \\
\left(\frac{\alpha}{a}\right)^4 + \left(\frac{\alpha}{a}\right)^2 &= \left(\frac{b}{a^2}\right)^2 \\
&\cdots \\
\left(\frac{\alpha}{a}\right)^{2^{32}} + \left(\frac{\alpha}{a}\right)^{2^{31}} &= \left(\frac{b}{a^2}\right)^{2^{31}}
\end{aligned}$$

Sum the equations above together, we have

$$\left(\frac{\alpha}{a}\right)^{2^{32}} + \frac{\alpha}{a} = Tr\left(\frac{b}{a^2}\right).$$

Because $x^2 + ax + b$ has no roots over $GF(2^{32})$, it is easy to prove that $Tr\left(\frac{b}{a^2}\right) = 1$. Thus,

$$\alpha^{2^{32}} + \alpha = a.$$

Therefore,

$$Truncate(H)a = Tr_{32}^{64}(H).$$

Since $a \neq 0$,

$$Truncate(H) = Tr_{32}^{64}(H)a^{-1}.$$

$\square$

By Lemma 1, Eqn.(2.1) can be written in a mathematical way, which is easier to analyze.

$$Tr_{32}^{64}\big((\sum_{i=0}^{n-1} P^{n-i}M_i + LEN)Q\big)a^{-1} + OTP.$$

2.3.3 EIA3

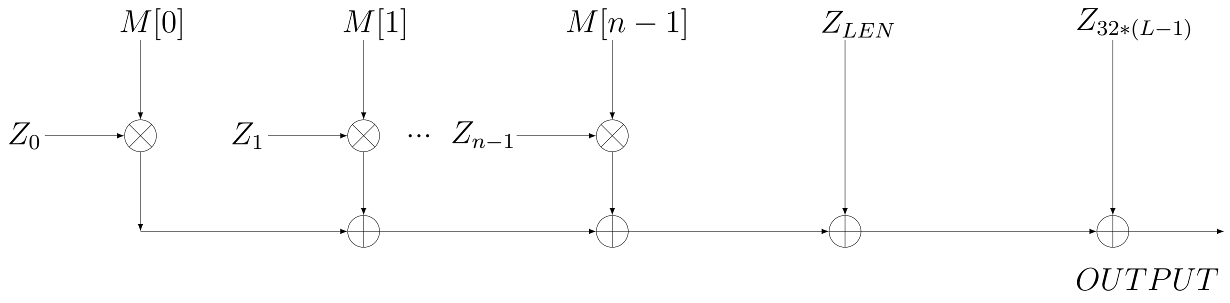


Figure 2.8: EIA3: based on ZUC

Originally, ZUC is not in the UMTS standard. It is added to the standard after the system migrates to LTE. The integrity protection based on ZUC is EIA3 [6]. EIA3 borrows the idea of Krawczyk [73], which uses shifted key streams to generate the MAC. In Figure 2.8, $M[i]$ is the i-th bit of message, and Z_i is the i-th word of the key stream generated by ZUC. Assume the key stream generated by ZUC is

$$Z = (Z[0], Z[1], \cdots, Z[32 * L - 1]),$$

where each $Z[i]$ is a bit. Then, Z_i is defined by

$$Z_i = (Z[i], Z[i+1], \cdots, Z[i+31]).$$

EIA3 pads every message with a"1". $Z_{32*(L-1)}$ is the mask to encrypt the intermediate tag. L is equal to $\lceil LENGTH/32 \rceil + 2$.

A mathematical expression of Figure 2.8 is given by

$$MAC(\mathbf{M}) = \underbrace{\sum_{i=0}^{n-1} M[i]z_i}_{part1} + \underbrace{z_{LENGTH} + z_{32*(L-1)}}_{part2},$$

The main observation of EIA3 is that Part 1 is a linear operation and Part 2 is a constant if the length is fixed.

2.4 Universal Hash Functions and APN Functions

We use the definitions of δ function and $universal_2$ hash function in Carter and Wegman's work [36] to illustrate the concept of universal hash function family.

Definition 1. *Let $\mathcal{H}$ be a hash function family. $\forall f \in \mathcal{H}$, $f : \mathcal{A} \mapsto \mathcal{B}$. Define $\delta_f(x,y)$ as follows.*

$$\delta_f(x,y) = \begin{cases} 1 & x \neq y \text{ and } f(x) = f(y) \\ 0 & otherwise \end{cases},$$

where $x, y \in \mathcal{A}$. Define $\delta_{\mathcal{H}}(x,y)$ using $\delta_f(x,y)$.

$$\delta_{\mathcal{H}}(x,y) = \sum_{f \in \mathcal{H}} \delta_f(x,y).$$

Definition 2. *Let $\mathcal{H}$ be a class of hash functions from $\mathcal{A}$ to $\mathcal{B}$. We say that $\mathcal{H}$ is $universal_2$ if $\forall x, y \in \mathcal{A}$, $\delta_{\mathcal{H}}(x,y) \leq |\mathcal{H}|/|\mathcal{B}|$. That is, $\mathcal{H}$ is $universal_2$ if no pair of distinct keys collide under more than $1/|\mathcal{B}|$-th of functions.*

Carter and Wegman explained the subscript "2" as the intention to emphasize that the definition constrains the behaviour of $\mathcal{H}$ only on pairs of elements in $\mathcal{A}$. Carter and Wegman defined the strongly universal$_n$ and strongly universal$_\omega$ hash function families in another paper [109].

Definition 3. *Suppose $\mathcal{H}$ is a set of hash functions, each element of $\mathcal{H}$ being a function from $\mathcal{A}$ to $\mathcal{B}$. $\mathcal{H}$ is strongly universal$_n$ if given any n distinct elements $a_0, \cdots, a_{n-1}$ of $\mathcal{A}$ and any n (not necessarily distinct) elements $b_0, \cdots, b_{n-1}$ of $\mathcal{B}$, then $|\mathcal{H}|/(|\mathcal{B}|^n)$ functions take a_0 to b_0, a_1 to b_1, etc. A set of hash functions is strongly universal$_\omega$ if it is strongly universal$_n$ for all values of n.*

Nevelsteen *et al.* formalized the definitions of *Almost Universal* (AU) hash function and *Almost Strongly Universal* (ASU) hash function based on the definitions proposed by Stinson [103] in their paper [90]. Besides, they also formalized the concept of *Almost Xor Universal* (AXU) hash function based on the definition in Krawczyk's paper [73].

Definition 4. *Let ϵ be any positive real number. An ϵ-almost universal family (or ϵ-AU family) $\mathcal{H}$ of hash functions from a set $\mathcal{A}$ to a set $\mathcal{B}$ is a family of functions from $\mathcal{A}$ to $\mathcal{B}$ such that for any distinct elements $x, x' \in \mathcal{A}$*

$$|\{h \in \mathcal{H} : h(x) = h(x')\}| = \delta_H(x, x') \le \epsilon \cdot |\mathcal{H}|.$$

Definition 5. *Let ϵ be any positive real number. An ϵ-almost strongly universal family (or ϵ-ASU family) $\mathcal{H}$ of hash functions from a set $\mathcal{A}$ to a set $\mathcal{B}$ is a family of functions from $\mathcal{A}$ to $\mathcal{B}$ such that*

- *for every $x \in \mathcal{A}$ and for every $y \in \mathcal{B}$, $|\{h \in \mathcal{H} : h(x) = y\}| = |\mathcal{H}|/|\mathcal{B}|$,*
- *for every $x_1, x_2 \in \mathcal{A}$ $(x_1 \neq x_2)$ and for every $y_1, y_2 \in \mathcal{B}(y_1 \neq y_2)$, $|\{h \in \mathcal{H} : h(x_1) = y_1, h(x_2) = y_2\}| \le \epsilon \cdot |\mathcal{H}|/|\mathcal{B}|$.*

Definition 6. *Let ϵ be any positive real number. An ϵ-almost XOR universal family (or ϵ-AXU family) $\mathcal{H}$ of hash functions from a set $\mathcal{A}$ to a set $\mathcal{B}$ is a family of functions from $\mathcal{A}$ to $\mathcal{B}$ such that for any distinct elements $x, x' \in \mathcal{A}$ and for any $b \in \mathcal{B}$*

$$|\{h \in \mathcal{H} : h(x) + h(x') = b\}| \le \epsilon \cdot |\mathcal{H}|.$$

Without ambiguity, we sometimes use AXU hash function to indicate the AXU hash function family throughout this thesis.

McGrew and Viega defined a class of message authentication code called *Almost Xor Universal MAC* (AUX-MAC) [82] based on AXU hash function family.

Definition 7. *A message authentication code is called ϵ-AXU MAC if it is defined as*

$$MAC = H(K, \overline{\mathbf{M}}) + R(N),$$

where K is the key; M is the message; N is the nonce; H is a ϵ-AXU hash function, and R is a random function.

When discussing the security of a message authentication code, Krawczyk proposed a concept called ϵ-otp-secure [73]. Before the definition, he proposed a secure model.

Secure model: Let $\overline{\mathbf{M}}$ be a message of length m authenticated with the tag $t = h(\overline{\mathbf{M}})+r$, where $h \overset{\$}{\leftarrow} \mathcal{H}$ and $r \overset{\$}{\leftarrow} \{0,1\}^n$. We say that an adversary that sees $\overline{\mathbf{M}}$ and t succeeds in breaking the authentication if it find $\overline{\mathbf{M}}$' and t', where $\overline{\mathbf{M}}$' is different than $\overline{\mathbf{M}}$ and $t' = h(\overline{\mathbf{M}'})+r$. We assume that the adversary knows the family of hash functions, but not the particular value of h or the pad r.

Definition 8. *A family $\mathcal{H}$ of hash functions is called ϵ-otp-secure if for any message M no adversary succeeds in the above scenario with probability larger than ϵ.*

Simmons defined two kinds of forgery attacks. Here we borrow the definition in [103].

Definition 9. *When an opponent places a new message $m' = (s', a')$, where s' is the source state, and a' is the authenticator, this is called* **impersonation forgery**. *When the opponent sees a message $m = (s, a)$ and changes it to a message $m' = (s', a')$ where $s \neq s'$, this is called* **substitution forgery**.

Note that the impersonation forgery means the opponent can forge a message-MAC pair without accessing to the oracle. The substitution forgery means the opponent accesses to the oracle once, and then forge a message-MAC pair.

The AXU property is quite similar to the definition of the APN function, which is defined as follows.

Definition 10. *Let $f(x) : \mathbb{F}_{2^n} \mapsto \mathbb{F}_{2^n}$. For any $a, b \in \mathbb{F}_{2^n}$, we denote*

$$\delta(a, b) = \#\{x \in \mathbb{F}_{2^n} : f(x) + f(x + a) = b\}.$$

If

$$\max_{a\neq 0, b\in\mathbb{F}_{2^n}} \delta(a, b) = 2,$$

the function $f(x)$ is called an almost perfect nonlinear (APN) function.

Thus, it is straightforward that the APN function can be applied to the AXU MAC.

Chapter 3

Linear Forgery Attack on EIA1

EIA1 has been proved to be a $2^{-32} + L \times 2^{-64}$-AXU MAC. Thus, the theoretical success rate of the substitution forgery attack is upper bounded by $2^{-32} + L \times 2^{-64}$. If we consider the security proof of the AXU MAC in Krawczyk's work [73], which will be discussed in the next chapter, there is a very strong assumption that the proof needs a one-time pad. However, as we all know, a one-time pad is not realistic in real systems. Therefore, this assumption has been criticized. For example, several researchers have proposed a concept called the robust authenticated-encryption scheme. According to Hoang, Krovetz and Rogaway, the "robust authenticated-encryption scheme [is] a new and very strong notion that implies protection of the privacy and authenticity of M and the authenticity of N and A, and must do so to the maximal extent possible even if nonces get reused ('misuse resistance')." [64]

Several recent works have focused on this property called "misuse resistance" [95]. Since the most significant misuse of AE schemes is the repetition of the nonce, most of misuse-resistant schemes are designed to provide some level of security when the nonce is repeated. The first scheme, called deterministic authenticated-encryption (DAE) [95], is proposed by Rogaway and Shrimpton. The very recent work is AEZ [64]. When misuse-resistant schemes are used improperly, the security levels of these schemes drop from the optimal levels to lower levels. Compared with misuse-resistant schemes, the ordinary schemes have no security at all when misuse occurs.

Since misuse can hardly be avoided, it is quite interesting to see what will happen to the ordinary schemes, when the nonce is repeated. Thus, in this chapter, we reveal some issues of the integrity protection of the 4G LTE system, when the nonce is repeated.

The linear structures of EIA1 and EIA3 enable us to forge a valid MAC if we know two

MACs generated by the same IV and IK. Note that the repetition of (IV, IK) is a kind of misuse. On top of such facts, we develop a method, by which we can forge up to 2^{32} valid MACs with two known valid MACs. Since we have 2^{32} valid MAC-message pairs, the probability of finding a pair with a valid MAC and valid message is quite significant. Statistically, there is more than one meaningful pair among those 2^{32} pairs. For brute-force attacks, the probability of finding the valid MAC of a message is $1/2^{32}$, but the message is not guaranteed to be valid (meaningful). Now the probability that we will get a meaningful MAC-message pair is increased to more than $1/2^{32}$. In fact, finding the parameter that can generate a meaningful pair is much easier in practice, because the messages usually have some specific structures, which may shrink the search space. In addition, such an attack aims not only at EIA1 and EIA3 but also at general polynomial MACs.

To prove that our linear forgery attack is doable, we create a scenario from which our attack can be launched. This scenario is based on the observation that the authentication of LTE is not really mutual, although it is claimed to be a mutual authentication. In EPS-AKA (the AKA procedure of LTE) only the server sends a challenge to a client. Checking the response of the challenge, the server can authenticate clients. Nevertheless, a client does not challenge the server. Thus, it can only authenticate the server by checking the MAC of the authentication vector. Such a protocol leaves a hole for replay attacks. Usually, replay attacks on EPS-AKA cannot obtain any information. However, because of our attack, the replay attack makes the forgery possible.

The rest of this chapter is organized as follows. Section 3.1 introduces how to synchronize the initial vector (IV) of the underlying cipher, and how to derive the integrity key (IK) in LTE. Section 3.2 presents certain security issues of EIA1 and EIA3, and proposes a forgery attack on EIA1 and EIA3. Since the attack makes use of the linearity of MACs, we call it a *linear forgery attack*. Section 3.3 describes a scenario from which the linear forgery attack can be launched in practice, and shows some experimental results obtained from the attack.

3.1 IV Synchronization Mechanism and Derivation of IK

EIA1, EIA2 and EIA3 synchronize IVs in the same way. As required by the EIA family, $COUNT-I$ and $FRESH$ are inputted as parameters to form IV of the underlying cipher. $FRESH$ is a random number transmitted by the network to the UE during the security mode set-up procedure. $COUNT-I$ is a counter that records how many times IK has already been used previously. This counter consists of two parts, RRC SQN and hyper frame number (HFN). RRC SQN is the sequence number of the radio resource control commands, and increases every time a package is sent, whether successfully or unsuccessfully. When RRC SQN overflows, HFN increases. $COUNT-I$ and $FRESH$ together are used to help the system prevent replay attacks.

$COUNT-I$ is written in the header of each package to synchronize the counters of the transmitter and receiver named $COUNTER_{tx}$ and $COUNTER_{rx}$, respectively. The transmitter uses $COUNTER_{tx}$ as the value of its $COUNT-I$, and then generates a key stream. When the receiver receives a package, the value of $COUNT-I$ is compared with the value of $COUNTER_{rx}$. If the value of $COUNT-I$ is greater than $COUNTER_{rx}$, the received $COUNT-I$ together with $FRESH$ are used to form IV, and the value of $COUNTER_{rx}$ is replaced by the received $COUNT-I$. Then, the key stream generated using this IV is used to verify the MAC. If the value of $COUNT-I$ is smaller than $COUNTER_{rx}$, this package will be treated as a repeated package and disregarded.

IK is derived in EPS-AKA. To prevent replay attacks, EPS-AKA needs SQN to identify those replayed messages. This SQN is different from RRC SQN mentioned before. RRC SQN is the sequence number of RRC commands, and SQN is the sequence number of the AKA procedure. At the beginning of each session (a session means the duration between two AKAs), RRC SQN will be set to an initial value. Compared with RRC SQN, SQN is used in the AKA procedure, and at the beginning of each session, SQN continues with the value of the last session. Therefore, both UEs (precisely USIMs) and the AuC record the SQN of the last session. In order to conduct the replayed AKA in our attack, we need to make SQN wrap around.

The documents of 3GPP do not specify the implementation of SQN, because it does not affect the interoperability. USIM and AuC belong to the same operator, which by itself can decide the implementation SQN. Thus, specification of SQN in the standard is not necessary. Instead, 3GPP provides only some suggestions and recommended implementation examples.

Here, we present one recommended implementation in TS33.102 [8], which is called *Profile 2*. SQN has two portions, SEQ and IND. Each subscriber has an individual SEQ counter on the AuC's side. On the subscriber's side, SEQ of the last session is stored in USIM. For the purpose of description, we call the value stored on the user's side SEQ_{MS}, and we call the SEQ stored on the AuC's side SEQ_{HE}. To store the value of SEQ, both AuC and USIM have an array, whose size is $2^{LEN(IND)}$, where $LEN(IND)$ represents the length of IND. Let us take the array on UE's side as an example to illustrate the verification of SQN. Denote the i-th slot of SEQ_{MS} as $SEQ_{MS}[i]$. After the received authentication vector passes the verification of MAC and SQN, the new SEQ goes into this array in the following way. Assume the the values of the received SEQ and IND are SEQ_0 and IND_0. SEQ_0 is written to the IND_0-th slot ($SEQ_{MS}[IND_0]$). In the next AKA procedure, assume the received SEQ and IND are SEQ_1 and IND_1. To verify SQN, USIM retrieves $SEQ_{MS}[IND_1]$ from the IND_1-th slot. If $SEQ_1 > SEQ_{MS}[IND_1]$ and $SEQ_1 < SEQ_{MS}[IND_1] + \Delta$, this SQN will be accepted, where Δ is the variable to prevent SQN from wrapping around. Otherwise, USIM will send an AUTS to resynchronize.

As mentioned in the introduction, AuC generates a batch of AVs and sends them to MME in each AKA procedure. Because of the traffic delay, those vectors may arrive at MME out of order. However, since USIM maintains an array to record SEQ, such disorder will not cause rejection of a valid AV. For example, $\{AV_i | 0 \leq i < n\}$ is received in the order of $\{AV_{i_0}, AV_{i_1}, \cdots, AV_{i_{n-1}}\}$. They have the same SEQ but different INDs. When USIM receives AV_i with SQN_i, it writes SEQ_i into the IND_i-th slot. In the next session, USIM receives AV_j with SQN_j, where $SEQ_j = SEsQ_i$. However, since $IND_j \neq IND_i$, and the value in IND_j-th slot should be smaller than SEQ_j (assuming no errors or attacks occur), then USIM can accept SQN_j. The length of IND affects how many AVs can be delivered from AuC to MME each time.

Profile 2 in Appendix C.3.2 of TS33.102 suggests that the lengths of SEQ and IND are 43-bit and 5-bit, respectively. The recommended value of Δ is 2^{28}.

3.2 Security Issue of EIA1 and EIA3

We present our work in this section. Compared with other works before, our attack is more practical. We need only two valid message-MAC pairs to forge another valid message-MAC pair.

3.2.1 Quasi-Linearity Property of EIA1 and EIA3

Let $\mathbf{M} = (M_0, M_1, \cdots, M_{n-1})$, where M_i is a 64-bit vector, treated as an element in $GF(q^2)$, which is defined by a primitive polynomial $t(x) = x^{64} + x^4 + x^3 + x + 1$. Let α be a root of $t(x)$ in $GF(q^2)$, and let $\beta = \alpha^{2^{32}+1}$. Then β is a primitive element of $GF(q)$, a subfield of $GF(q^2)$. The minimal polynomial of α over $GF(q)$ is given by

$$t_1(x) = x^2 + ux + v,$$

where $u = \beta^{17}$, $v = \beta$. Then each element in $GF(q^2)$ can be represented as $a + b\alpha$, $a, b \in GF(q)$. We define

$$f(\mathbf{M}, P) = \sum_{i=1}^{n} M_{n-i}P^i, for\ P \in GF(q^2),\ M_i \in GF(q^2).$$

Property 1. *For any $\lambda \in GF(q)$, a subfield of $GF(q^2)$,*

$$\begin{aligned}\mathbf{M}_1 &= (M_{1,0}, M_{1,1}, \cdots, M_{1,n-1})\\ \mathbf{M}_2 &= (M_{2,0}, M_{2,1}, \cdots, M_{2,n-1}),\end{aligned}$$

then

$$f(\boldsymbol{M}_1 + \boldsymbol{M}_2, P) = f(\boldsymbol{M}_1, P) + f(\boldsymbol{M}_2, P) \tag{3.1}$$

$$f(\lambda\boldsymbol{M}, P) = \lambda f(\boldsymbol{M}, P). \tag{3.2}$$

Proof. According to the definition of $f(\mathbf{M}, P)$, we have

$$\begin{aligned}f(\mathbf{M}_1 + \mathbf{M}_2, P) &= \sum_{i=1}^{n}(M_{1,n-i} + M_{2,n-i})P^i\\ &= \sum_{i=1}^{n} M_{1,n-i}P^i + \sum_{i=1}^{n} M_{2,n-i}P^i\\ &= f(\mathbf{M}_1, P) + f(\mathbf{M}_2, P).\\ f(\lambda\mathbf{M}, P) &= \sum_{i=1}^{n}(\lambda M_{n-i})P^i\\ &= \lambda\sum_{i=1}^{n} M_{n-i}P^i = \lambda f(\mathbf{M}, P).\end{aligned}$$

Thus, the assertions are true. $\square$

We have the MAC of $\mathbf{M}$ generated by EIA1 as

$$MAC(\mathbf{M}) = [Q \cdot f(\mathbf{M}, P)]_{0..31} + [Length \cdot Q]_{0..31} + OTP$$

Let $Q \cdot f(\mathbf{M}, P) = a + b\alpha$, $Length \cdot Q = c + d\alpha$, a, b, c, and $d \in GF(q)$. From the MAC generation of EIA1 introduced in Section 2.3 and Property 1, the following result follows immediately.

Property 2. *For any $\lambda \in GF(q)$,*

$$\begin{aligned} MAC(\mathbf{M}) &= a + c + OTP && (3.3)\\ MAC(\lambda\mathbf{M}) &= \lambda a + c + OTP. && (3.4)\end{aligned}$$

EIA3 can be equivalent to a polynomial evaluation based MAC. Then the linear forgery attack can be directly applied to it.

3.2.2 Linear Forgery Attack Algorithm

Assume that we can make three queries to a MAC oracle to obtain MACs of the messages $\mathbf{M}_i$, for $i = 1, 2, 3$, under the same IV. Let

$$Q \cdot f(\mathbf{M}_i, P) = a_i + b_i\alpha,\ a_i,\ b_i \in GF(q). \tag{3.5}$$

Theorem 1. *Let (i, j, k) be a permutation of $(1, 2, 3)$. For any $\lambda \in GF(q)$*

$$MAC(\mathbf{M}_{new}) = \lambda(MAC(\mathbf{M}_i) + MAC(\mathbf{M}_j)) + MAC(\mathbf{M}_k) \tag{3.6}$$

which is a valid MAC value of the message

$$\mathbf{M}_{new} = \lambda(\mathbf{M}_i + \mathbf{M}_j) + \mathbf{M}_k.$$

Proof. Since the proofs for the other cases are similar, we give a proof only for $(i, j, k) = (1, 2, 3)$. In order to prove (3.6), we compute both sides of (3.6). According to Properties 1 and 2

$$MAC(\mathbf{M}_{new}) = \lambda(a_1 + a_2) + a_3 + c + OTP. \tag{3.7}$$

On the other hand,

$$\begin{aligned} &\lambda(MAC(\mathbf{M}_1) + MAC(\mathbf{M}_2))) + MAC(\mathbf{M}_3)\\ &\quad = \lambda(a_1 + c + OTP + a_2 + c + OTP) + a_3 + c + OTP\\ &\quad = \lambda(a_1 + a_2) + a_3 + c + OTP. && (3.8)\end{aligned}$$

The assertion follows from (3.7) and (3.8). □

From (3.6), we have the following corollary.

Corollary 1. *Let (i, j) be a permutation of $(1, 2)$, and $k \in (1, 2)$. For any $\lambda \in GF(q)$, (3.6) is true. In other words, if we have the valid MACs from two queries, then*

$$\begin{aligned} &MAC(\lambda(\boldsymbol{M}_1 + \boldsymbol{M}_2) + \boldsymbol{M}_1) \\ &\quad = \lambda(MAC(\boldsymbol{M}_1) + MAC(\boldsymbol{M}_2)) + MAC(\boldsymbol{M}_1) \\ &MAC(\lambda(\boldsymbol{M}_1 + \boldsymbol{M}_2) + \boldsymbol{M}_2) \\ &\quad = \lambda(MAC(\boldsymbol{M}_1) + MAC(\boldsymbol{M}_2)) + MAC(\boldsymbol{M}_2) \end{aligned}$$

are valid.

From Corollary 1, we need only two valid MACs to forge a new one. In practice, we can reduce the number of queries by applying Corollary 1. Obtaining two valid MACs generated by the same IV is much easier than obtaining three.

The algorithm that can forge a valid MAC using two known valid MACs is shown in Algorithm 1, where $find\lambda()$ is a function that returns a $\lambda \in GF(q)$ such that either $\lambda(\mathbf{M}_1 + \mathbf{M}_2) + \mathbf{M}_1$ or $\lambda(\mathbf{M}_1 + \mathbf{M}_2) + \mathbf{M}_2$ is a valid message. How to find λ, such that the message is also valid, is discussed in Section 3.2.3.

Algorithm 1: Linear forgery

```
Data: two messages $\mathbf{M}_1$, $\mathbf{M}_2$, and the MACs of these two messages $MAC(\mathbf{M}_1)$,
      $MAC(\mathbf{M}_2)$
Result: one message and its valid MAC
$\lambda = find\lambda();$
temp = $\lambda(\mathbf{M}_1 + \mathbf{M}_2)$;
if temp + $\boldsymbol{M}_1$ is a valid message then
    $\mathbf{M}_{new}$ = temp + $\mathbf{M}_1$;
    $MAC(\mathbf{M}_{new}) = \lambda(MAC(\mathbf{M}_1) + MAC(\mathbf{M}_2)) + MAC(\mathbf{M}_1)$;
else
    $\mathbf{M}_{new}$ = temp + $\mathbf{M}_2$;
    $MAC(\mathbf{M}_{new}) = \lambda(MAC(\mathbf{M}_1) + MAC(\mathbf{M}_2)) + MAC(\mathbf{M}_2)$;
end
return $MAC(\boldsymbol{M}_{new})$ and $\boldsymbol{M}_{new}$;
```

Remark 1. *The length of the MAC generated by EIA family is 32-bit. An attacker can randomly select 32 bits to forge the MAC of a message; the success probability is $1/2^{32}$.*

However, if the attacker can make two queries to obtain two valid MACs, then he can forge 2^{32} *messages with valid MACs. In Section 3.3, we will demonstrate how the attacker can obtain two valid MACs in practice.*

3.2.3 How To Find λ

We randomly pick a λ, then we can forge a MAC of a message. However, this message may not be a meaningful message of a protocol. Thus, the problem is how to find a λ that can generate the MAC of a meaningful message.

Usually, in a real system, the two queried messages have certain relationship, which let us easily find a λ. Take counter check message as an example. Two counter check messages have very similar structures. Therefore, most bits in the XOR of two counter check messages are zeros. The only bits that we need to consider are those nonzero bits, which are minority in the result of XOR.

Moreover, even we cannot find the valid message, our linear forgery attack can still cause Denial-of-Service (DoS) attacks. Because the MAC is valid, every time the receiver must do the decoding and then finds the message is not well formatted. Note that simply flipping bits can also be a DoS attack, by which the server will stop after verifying the MAC. However, our attack can make the server keep computing until the message is decoded. Therefore, the computational resource will be occupied by verifying and decoding. Compared with flipping bits, our attack brings more computation to the server.

3.3 Application

In this section, we design a scenario, in which the same IV and IK will occur twice. Because of the repetition, our linear forgery attack can be launched to get the valid MAC.

3.3.1 A Scenario of Fixing IV

The procedure that make the same (IV, IK) combination happen twice in two consecutive sessions is essentially a replay attack. Without ambiguity, we call this procedure a replay attack. Figure 3.1 demonstrates the replay attack by which we can make the same RRC SQN and IK occur twice in two different sessions. The repetition of IK leads to the consequence that FRESH is repeated in two sessions. Since IV is composed of RRC SQN

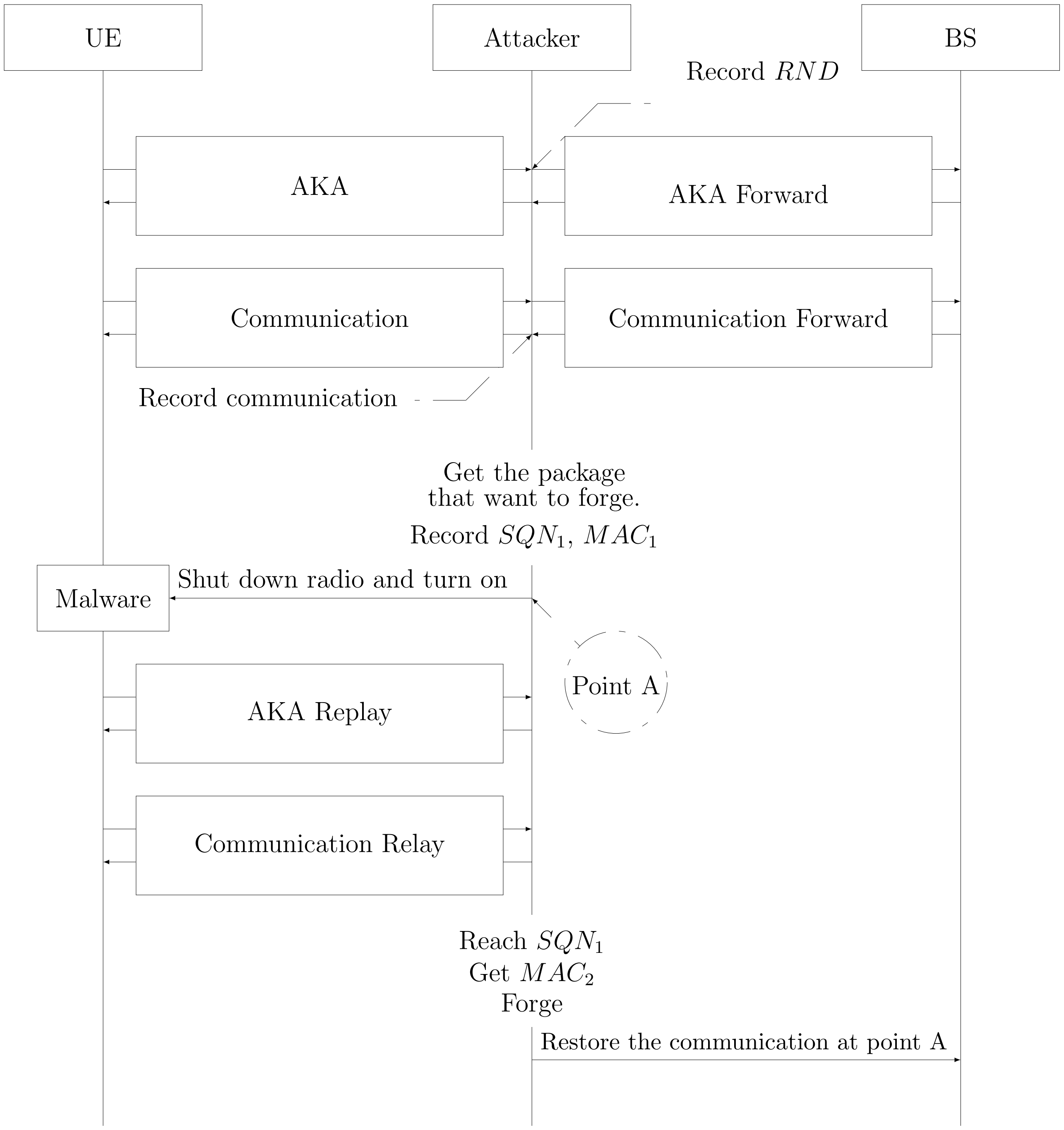


Figure 3.1: Fixing IV

and FRESH, IV is repeated as well. The same (IV, IK) combination in two consecutive sessions determines the same initial states of the underlying stream cipher, which will generate the same key stream in two different sessions, consequently. The preconditions of this replay attack are: (1) we can set up the man-in-the-middle (MITM), and (2) there is a malware on the phone that can shut down and turn on the radio. Perez *et al.* [94] show that Condition (1) is applicable. Condition (2) is also easy to be satisfied. We can choose the Android smart phone to be our target because Android is an open platform, and is widely used around the world.

The whole replay attack procedure is described as follows. First, the MITM attacker records all user data messages and control messages, including the authentication and key agreement messages. When this attacker observes the package he wants to forge, he shuts down the radio of the victim and then turns it on. The MITM attacker uses the recorded AKA messages to conduct a replay attack. In the AKA protocol, mobile devices are not required to verify whether the random number has been received before or not. They only check the freshness of SQN. However, in some cases, we can make SQN wrap around (the details are shown in full version of our paper [110]). Thus, the victim believes it is talking with the real base station. Notice that the EPS-AKA is claimed to be mutually authenticated. The UE proves its identity to the MME by replying to the challenge from the MME. However, the UE does not send any challenges to the MME. The UE can verify the MME only by checking the MAC of the authentication vector. The random number in the authentication vector can make sure each authentication vector is unique. However, the UE cannot record all random numbers it received before. This enables the replay attack. Such attack makes the UE accept the fake MME. Generally, the attacker can get nothing from the replay attack, because he still cannot get the key. But in our case, we do not care about the key. The only thing we care about is the RRC SQN. As long as we get two identical RRC SQNs with the same IK, we can launch our linear forgery attack.

Upon the victim accepts the random number, it generates the same IK, which is also used in the session suspended by the attacker. When the victim believes the attacker is the real MME, it begins to send packages. The attacker replies with the previously recorded packages. The victim may accept or reject those packages, but it does not matter, because the only target for the attacker is to increase the victim's counter until the RRC SQN reaches the recorded value.

As long as we get the sequence number that we want, the MITM attacker applies our linear forgery attack to forge a valid MAC of the package. This forged package together with the forged MAC will be forwarded to the real base station. Since the MAC will pass the verification, this package will be accepted.

3.3.2 Counter Check Message

A practical application of our linear forgery attack is forging counter check messages. Because the integrity of the user plane is not protected in LTE, counter check messages are sent from the MME to UE to check the number of transmitted data packages. The MME writes the most significant s bits of its counter in the counter check message and sends it to UE, who compares its own counter with the received value. Then, UE sends the most significant s bits of its counter back to the MME. Upon receiving the value sent back by UE, the MME computes the difference between the received value and its counter. If the difference is not acceptable, the MME releases the connection. This procedure is shown in Figure 3.2. Chen *et al.* [38] present more details about the counter check message.

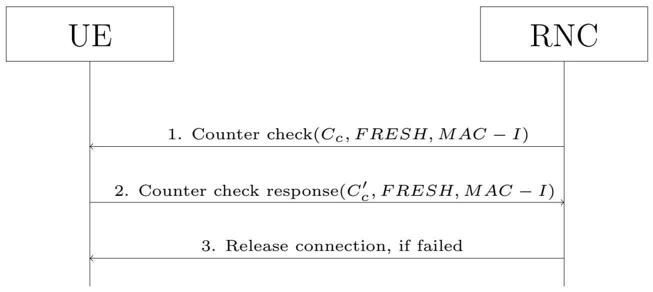

UE
RNC
1. Counter check($C_c, FRESH, MAC-I$)
2. Counter check response($C'_c, FRESH, MAC-I$)
3. Release connection, if failed

Figure 3.2: Counter Check Message.

Attackers may sometimes want to insert some data (malicious codes, advertisement, etc.) into the user data stream. If the counter check message is conducted correctly, the MME will find out the insertion and release the connection. In this point of view, attackers need to forge a counter check message to deceive the MME.

3.3.3 Launching Attack

We assume that the MAC-I in Figure 3.2 is generated by EIA1 or EIA3. IVs of EIA1 and EIA3 are composed of two portions: the least significant four bits represent the RRC SQN, and the other twenty-eight bits represent the HFN. The RRC SQN is increased by one each time when an RRC signal is sent whether successfully or unsuccessfully. If there is an overflow of the RRC SQN, the HFN is increased by one.

Attacking scenario:

1. MME sends a counter check message to the MITM attacker.
2. The MITM attacker forwards this message to the UE, and gets the reply form the UE with MAC_1.
3. The MITM attacker applies the attack we mentioned above, and gets MAC_2.
4. The MITM attacker forges $\lambda(MAC_1+MAC_2)+MAC_1$ or $MAC_2+\lambda(MAC_1+MAC_2)$, then forwards to the real base station.
5. The MME accepts the difference, and continues communicating with the MITM attacker.
6. The MITM attacker can continue to forward messages between the MME and the UE without being detected by the MME.

This process has a disadvantage that the connection between the MITM attacker and the MME may be time-out during the forgery process. So such attack can forge only the counter check message that is sent not too long after powering up.

3.3.4 Experimental Results

In order to test our attack, we generate two counter check messages, which consist of are two counters as shown in Table 3.1. The RRC commands of these two packages are listed in Table 3.2.

Table 3.1: Counter Check Messages

Message	Identity	UCounter	DCounter
$\mathbf{M}_1$	10	258	257
	50	260	259
$\mathbf{M}_2$	10	259	258
	50	261	260

Table 3.2: Counter Check Messages in Hex

Message	RRC command
$\mathbf{M}_1$	0x30 0x27 0xA1 0x25 0xA4 0x23 0xA0 0x21 0xA0 0x1F 0x80 0x01 0x1E 0xA1 0x1A 0x30 0x0B 0x80 0x01 0x0A 0x81 0x02 0x01 0x02 0x82 0x02 0x01 0x01 0x30 0x0B 0x80 0x01 0x32 0x81 0x02 0x01 0x04 0x82 0x02 0x01 0x03
$\mathbf{M}_2$	0x30 0x27 0xA1 0x25 0xA4 0x23 0xA0 0x21 0xA0 0x1F 0x80 0x01 0x1E 0xA1 0x1A 0x30 0x0B 0x80 0x01 0x0A 0x81 0x02 0x01 0x03 0x82 0x02 0x01 0x02 0x30 0x0B 0x80 0x01 0x32 0x81 0x02 0x01 0x05 0x82 0x02 0x01 0x04

Forgery Procedure

XOR of these two messages gives

0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01
0x00 0x00 0x00 0x03 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00
0x07

Then chose λ =0x1B, $\lambda(\mathbf{M}_1 + \mathbf{M}_2)$ is

0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x1B
0x00 0x00 0x00 0x2D 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x1B 0x00 0x00 0x00
0x41

Finally, we get the message $\mathbf{M}_{new} = \mathbf{M}_2 + \lambda(\mathbf{M}_1 + \mathbf{M}_2)$

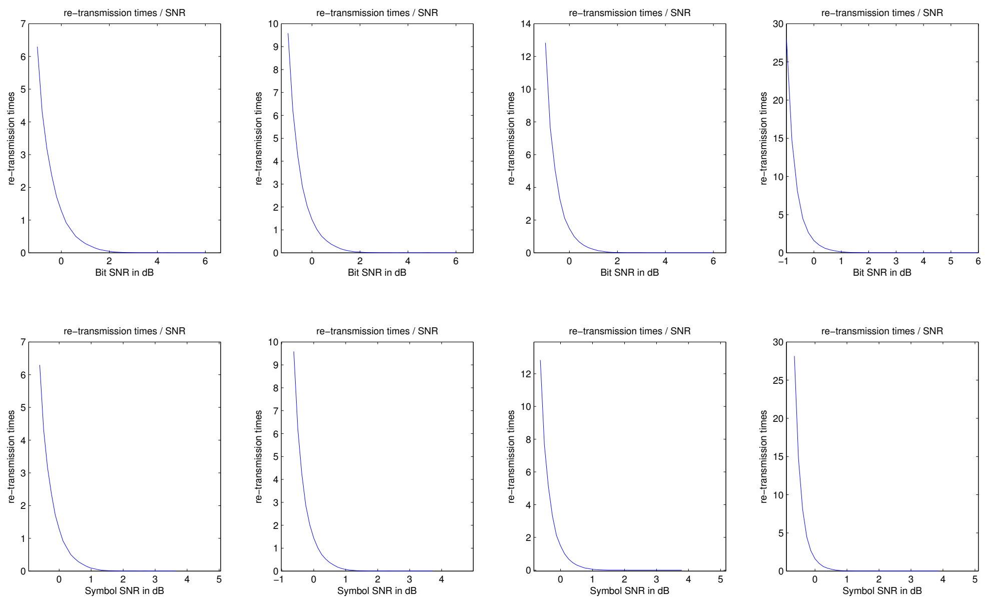


Figure 3.3: Retransmission times / SNR

```
0x30 0x27 0xA1 0x25 0xA4 0x23 0xA0 0x21
0xA0 0x1F 0x80 0x01 0x1e 0xa1 0x1A 0x30
0x0B 0x80 0x01 0x0A 0x81 0x02 0x01 0x18
0x82 0x02 0x01 0x2F 0x30 0x0B 0x80 0x01
0x32 0x81 0x02 0x01 0x1E 0x82 0x02 0x01
0x45
```

This forged message is represented in the binary form, whose XML form is shown in Figure 3.4. Obviously, the message in Figure 3.4 still contains the counter values of the bearers 10 and 50. In the forged counter check message, the uplink and downlink counter values of the bearer 10 are 577 and 531, which are greater than the original values. The uplink and downlink counter values of the bearer 50 are increased to 274 and 352 as well. Not only the counters of both bearers are incremented as what attackers want, but also the counter check message can pass the MAC verification. Therefore, attackers can successfully forge a valid package, in which the value of each counter is increased, and the MAC of the message is valid. The forgery leads to the consequent that if the attacker inserts some packages, the MME is unaware of the insertion.

Timing of Attack

Turning the radio off and on usually costs three to six seconds. If the connection is not time-out within this duration, our attack can be launched successfully. No specification of this time-out duration is suggested in the LTE standard, which means the duration is decided by network operators. We cannot measure this duration by experiments, because analysis of public communications is forbidden by the law. However, since RRC commands are transmitted in an ARQ fashion, we can use the retransmitting time to estimate the lower bound of the time-out duration. We simulate an Additive White Gaussian Noise (AWGN) channel, and choose the modulation scheme to be the Quaternary Phase Shift Keying (QPSK). The result is shown in Figure 3.3. Each column is corresponding to a set of coding parameters. The top row shows the bit Signal to Noise Ratio (SNR), while the bottom row represents the symbol SNR.

Figure 3.3 indicates that when the bit SNR or symbol SNR is around -0.2dB, the base station needs to transmit a message three times on average to ensure that the user can receive the transmitted message. When UEs are inside a building, SNR may even be smaller than -0.2dB. To make sure all users can get services, the time-out duration must be longer than three-time retransmission. This duration is long enough to give attackers a chance to conduct the attack.

```
<DL-DCCH-Message>
 <message>
  <counterCheck>
   <r3>
    <counterCheck-r3>
     <rrc-TransactionIdentifier>30</rrc-TransactionIdentifier>
     <rb-COUNT-C-MSB-InformationList>
      <RB-COUNT-C-MSB-Information>
       <rb-Identity>10</rb-Identity>
       <count-C-MSB-UL>577</count-C-MSB-UL>
        <count-C-MSB-DL>531</count-C-MSB-DL>
      </RB-COUNT-C-MSB-Information>
      <RB-COUNT-C-MSB-Information>
       <rb-Identity>50</rb-Identity>
       <count-C-MSB-UL>274</count-C-MSB-UL>
       <count-C-MSB-DL>352</count-C-MSB-DL>
      </RB-COUNT-C-MSB-Information>
     </rb-COUNT-C-MSB-InformationList>
    </counterCheck-r3>
   </r3>
  </counterCheck>
 </message>
</DL-DCCH-Message>
```

Figure 3.4: Decoding result

3.4 Summary

In this chapter, we propose a method called linear forgery attack, whereby two known valid message-MAC pairs generated by the same IV and IK, attackers can forge 2^{32} valid message-MAC pairs. Compared with the random guessing or birthday attacks, our attack guarantees that the forged message-MAC pairs can pass the verification. Moreover, among those 2^{32} valid pairs, very likely, there exist some meaningful messages, which will not be discarded by the receiver because they are meaningless messages. In a real environment, we can easily find a meaningful message-MAC pair. This chapter demonstrates an example that generates a meaningful pair by forging a counter check message. We also develop an attack scenario that makes the same IV and IK occur twice. This scenario enables our linear forgery attack in practice.

To prevent our linear forgery attack, the structures of EIA1 and EIA3 need to be changed such that either the message is involved in generating the key stream or the MAC is generated in a nonlinear fashion. We can hardly find a way to avoid linear structures without compromising efficiency. So far, security and efficiency of EIA1 and EIA3 are trade-off.

Chapter 4

Security Analysis and Efficient Implementation of EIA1

In Krawczyk's paper[73], he proved that if the MAC was constructed by adding an ϵ-AXU hash and a one-time pad together, the success probability of substitution forgery attacks against this MAC is upper bounded by ϵ. However, in his proof, he directly replaced a conditional probability with an unconditional probability without any explanation. This proof left a gap between the ϵ-AXU property and the substitution forgery probability. Since the security of all AXU MAC constructions is proved using Krawczyk's theorem in this paper, his proof needs to be fixed in order to fix the security proofs of all AXU MACs.

As mentioned in the previous chapter, EIA1, as an AXU MAC, has its security proof. However, attacks against EIA1, such as the linear forgery attack, are found, because in Krawczyk's proof, he required a one-time pad, which is impractical. Since no one-time pad exists in the real world, usually it is replaced by a stream cipher or a block cipher in counter mode, which can be misused. Because of the misuse, the requirement is not satisfied, and the MAC is not information theoretically secure anymore. Therefore, the security of EIA1 (as well as other AXU MACs) should be examined under different security models.

The official implementation of EIA1 is published on ETSI's website. The latest version is V2.1. It is quite confusing that for the same message, V1.1 and V2.1 of EIA1 generate different tags. Moreover, the algorithm is implemented in a very inefficient way.

In this chapter, we fix the gap between the AXU property and the substitution forgery probability. Then, we present the security proof of EIA1 under different attack models. The security bound proposed in the evaluation report of EIA1 is a special case of our analysis. After the security proof, we point out some implementation flaws of EIA1, both V1.1 and

V2.1, published on the ETSI's website. We optimized the official implementation to get an optimized version of $f9$ function, which is called $f9_{opt}$. We also use an efficient polynomial evaluation method, which is analog to the fast Fourier transform, to improve the efficiency. Compared with Horner's rule, which is used in the official implementation, our method reduces the number of multiplications over finite field dramatically. After that, we show the improvement by the experiment results. The results show that our optimized code is much faster than the official implementation, and our polynomial evaluation method is better than Horner's Rule.

The remaining part of this chapter is organized as follows. Sections 4.1 fixes the proof of AXU MAC and assesses the security of EIA1 under different models. Section 4.2 discusses the improvement of EIA1, and shows the improvement by the experiment. The last section concludes the chapter.

4.1 Security Analysis of EIA1

In the evaluation report of EIA1 [1], it was claimed to be an $L \times 2^{-64} + 2^{-32}$-AXU MAC. Then the authors indicated the substitution forgery probability was upper bounded by $L \times 2^{-64} + 2^{-32}$. This idea is quite similar to the proof of Lemma 5 in McGrew and Viega's paper [82]. In Krawczyk's paper [73], he theoretically proved that ϵ-AXU MAC is resistant to substitution forgery attacks. However, Krawczyk directly replaced a conditional probability with an unconditional probability, which made the proof imprecise. In this section, we show the conditional probability is equal to the unconditional probability. The equality fixes Krawczyk's proof. Then, we analyze the security of EIA1 under four modes. Note that the security analysis in the evaluation report of EIA1 is only a subset of our results.

4.1.1 AXU Hash Is Secure

McGrew and Viega proved GCM is an AXU-MAC by counting the number of roots of a polynomial. The security proof of EIA1 has the same idea. The theory behind those proofs is a theorem proposed by Krawczyk in 1994.

Theorem 2. *[73] A necessary and sufficient condition for a family $\mathcal{H}$ of hash functions to be ϵ-otp-secure is that*

$$\forall M_1 \neq M_2, c, Pr_h[h(M_1) + h(M_2) = c] \leq \epsilon.$$

Krawczyk argued that if the attacker cannot find two message-tag pairs, such that the summation of these two tags equals a certain value with very high probability, this MAC is secure. However, in his proof of Theorem 2, he directly let the substitution forgery probability equal $Pr_h[h(M_1)+h(M_2)=c]$, which is the impersonation forgery probability. But under the *chosen-plaintext-attack* (CPA) model, the adversary has the ability to choose any plaintext based on the previous queries [71]. Thus, the substitution forgery probability should be $Pr_h[h(M_1)+h(M_2)=c|h(M_1)+R=t_1]$, where R is a random mask, under the CPA model. Note that there are three random variables in this probability, $h(M_1)$, $h(M_2)$ and R. From the probability theory, we know $Pr_h[h(M_1)+h(M_2)=c|h(M_1)+R=t_1]$ does not necessarily equal $Pr_h[h(M_1)+h(M_2)=c]$. To fix Krawczyk's proof, we prove the following lemma and theorem.

Lemma 2. *Let X, Y and R be random variables over $\mathbb{F}_2^m$, and denote $N=2^m$. R is independent from both X and Y, and uniformly distributed. Then,*

$$\begin{aligned} Pr[X+R=t] &= \frac{1}{N}, \text{ for any constant } t\in\mathbb{F}_2^m, && (4.1)\\ Pr[X+Y=c, X+R=t] &= \frac{1}{N}Pr[X+Y=c], \text{ for any } c \text{ and } t\in\mathbb{F}_2^m. && (4.2)\end{aligned}$$

Proof. Eqn. (4.1) equals

$$\sum_{r\in\mathbb{F}_2^m} Pr[X=r+t|R=r]Pr[R=r].$$

Since R is uniformly distributed, $Pr[R=r]=1/N$. Then,

$$\text{Eqn. (4.1)} = \frac{1}{N}\sum_{r\in\mathbb{F}_2^m} Pr[X=r+t|R=r]$$

Because r runs through every element in $\mathbb{F}_2^m$ and R is independent from X, then

$$\sum_{r\in\mathbb{F}_2^m} Pr[X=r+t|R=r]=1.$$

Thus,

$$\text{Eqn. (4.1)} = \frac{1}{N}.$$

From the proof above, Eqn. (4.2) equals

$$\frac{1}{N}\sum_{r\in\mathbb{F}_2^m} Pr[X+Y=c, X=r+t].$$

Because r runs through every element in $\mathbb{F}_2^m$, event $X + Y = c, X = r + t$ runs through all the possible $(x, y) \in \mathbb{F}_2^m \times \mathbb{F}_2^m$, such that $x + y = c$. Thereby,

$$\sum_{r \in \mathbb{F}_2^m} Pr[X + Y = c, X = r + t] = Pr[X + Y = c].$$

Thus,

$$\text{Eqn. (4.2)} = \frac{1}{N} Pr[X + Y = c].$$

This completes the proof. □

Theorem 3. *Let $\mathcal{H} = \{h | h : \mathbb{F}_2^* \mapsto \mathbb{F}_2^m\}$ be a hash function family, and $R \in \mathbb{F}_2^m$ be a random variable, which is uniformly distributed in $\mathbb{F}_2^m$.*

$$\begin{aligned} &Pr_h[h(M_1) + h(M_2) = c] \\ &\quad = Pr_h[h(M_1) + h(M_2) = c | h(M_1) + R = t_1]. \end{aligned}$$

Proof.

$$\begin{aligned} &Pr_h[h(M_1) + h(M_2) = c | h(M_1) + R = t_1] \\ &\quad = \frac{Pr_h[h(M_1) + h(M_2) = c, h(M_1) + R = t_1]}{Pr_h[h(M_1) + R = t_1]}. \end{aligned}$$

By Lemma 2

$$\begin{aligned} &Pr_h[h(M_1) + h(M_2) = c | h(M_1) + R = t_1] \\ &\quad = Pr_h[h(M_1) + h(M_2) = c]. \end{aligned}$$

□

From Theorem 3, Krawczyk's proof of Theorem 2 still holds for all AXU-MACs.

4.1.2 Security Analysis of EIA1

To thoroughly analyze the security of EIA1, we need to examine it under different models. We want to remark that the original security analysis of EIA1 is a subset of our results.

Let $\overline{\mathbf{M}}_i$ denote the i-th message and T_i denote the tag of $\overline{\mathbf{M}}_i$. LEN_i is the number of blocks of the message $\overline{\mathbf{M}}_i$. $\overline{\mathbf{M}}_i(x)$ is the polynomial, whose coefficients are the blocks of message $\overline{\mathbf{M}}_i$. P_i, Q_i and OTP_i are three random numbers generated by SNOW3G in the i-th query. Without ambiguity, if P_i, Q_i and OTP_i are constants in each query, we simply use P, Q and OTP without the subscript. There exists an oracle that accepts at most n queries, which are either generation or verification queries. By generation queries, attackers can send selected messages to the oracle, which sends back the tag of each queried message. Attacker can also query the oracle with verification queries to verify the validity of any given message-tag pairs. Attackers must forge a message-tag pair with at most n queries. Let us define four attack models as follows.

- $\mathcal{M}_0$: Adversaries get $(\overline{\mathbf{M}}_0, T_0)$ and forge $(\overline{\mathbf{M}}_1, T_1)$ with the same (P, Q, OTP). This is the substitution forgery by Simmons' definition.

- $\mathcal{M}_1$: Adversaries have the ability to get $(\overline{\mathbf{M}}_0, T_0), \cdots, (\overline{\mathbf{M}}_{n-1}, T_{n-1})$, where $\overline{\mathbf{M}}_i \neq \overline{\mathbf{M}}_j$ for $i \neq j$. For each $(\overline{\mathbf{M}}_i, T_i)$, $0 \leq i \leq n-1$, $T_i = Tr_{32}^{64}\left((\overline{\mathbf{M}}_i(P) + LEN_i)Q\right)a^{-1} + OTP$. Adversaries try to forge a pair $(\overline{\mathbf{M}}_n, T_n)$, such that
$T_n = Tr_{32}^{64}\left((\overline{\mathbf{M}}_n(P) + LEN_n)Q\right)a^{-1} + OTP$.

- $\mathcal{M}_2$: Adversaries have the ability to get $(\overline{\mathbf{M}}_0, T_0), \cdots, (\overline{\mathbf{M}}_{n-1}, T_{n-1})$, where $\overline{\mathbf{M}}_i \neq \overline{\mathbf{M}}_j$ for $i \neq j$. For each $(\overline{\mathbf{M}}_i, T_i)$, $0 \leq i \leq n-1$, $T_i = Tr_{32}^{64}\left((\overline{\mathbf{M}}_i(P_i) + LEN_i)Q_i\right)a^{-1} + OTP_i$. If $i \neq j$, $(P_i, Q_i, OTP_i) \neq (P_j, Q_j, OTP_j)$ with very high probability. Adversaries try to forge a pair $(\overline{\mathbf{M}}_n, T_n)$, such that $\exists 0 \leq i \leq n-1$
$T_n = Tr_{32}^{64}\left((\overline{\mathbf{M}}_n(P_i) + LEN_n)Q_i\right)a^{-1} + OTP_i$.

- $\mathcal{M}_3$: Adversaries have the ability to get $(\overline{\mathbf{M}}_0, T_0), \cdots, (\overline{\mathbf{M}}_{n-1}, T_{n-1})$, where $\overline{\mathbf{M}}_i \neq \overline{\mathbf{M}}_j$ for $i \neq j$. For each $(\overline{\mathbf{M}}_i, T_i)$, $0 \leq i \leq n-1$, $T_i = Tr_{32}^{64}\left((\overline{\mathbf{M}}_i(P_i) + LEN_i)Q_i\right)a^{-1} + OTP_i$. If $i \neq j$, $(P_i, Q_i, OTP_i) \neq (P_j, Q_j, OTP_j)$ with very high probability. Adversaries try to forge a pair $(\overline{\mathbf{M}}_n, T_n)$, such that
$T_n = Tr_{32}^{64}\left((\overline{\mathbf{M}}_n(P_n) + LEN_n)Q_n\right)a^{-1} + OTP_n$, where (P_n, Q_n, OTP_n) is different from (P_i, Q_i, OTP_i), for $0 \leq i \leq n-1$, with very high probability.

Security under $\mathcal{M}_0$

The bound in the evaluation report holds for EIA1 in this case. However, in the report, the authors did not give the explicitly proof of the forgery probability. We present our proof here for the purpose of this paper.

Proof. Let us use $M_{L_1}^1(x)$ and $M_{L_2}^2(x)$ denote the polynomial whose coefficients are the blocks of $\overline{\mathbf{M}}_1$ and $\overline{\mathbf{M}}_2$ respectively. The corresponding degrees of each polynomial are L_1

and L_2. $A = \{R \mid R \in GF(q^2), Tr_{32}^{64}(R)a^{-1} = \delta\}$. Therefore,

$$\begin{aligned}
&Pr[MAC(\overline{\mathbf{M}}_1) + MAC(\overline{\mathbf{M}}_2) = \delta] \\
&\qquad = Pr[(M_{L_1}^1(P) + LEN_1)Q + (M_{L_2}^2(P) + LEN_2)Q \in A].
\end{aligned}$$

Denote $LEN = LEN_1 + LEN_2$ and $M_L(P) = M_{L_1}^1(P) + M_{L_2}^2(P)$, where $L = \max(L_1, L_2)$. Then the above equation can be written as

$$Pr[(M_L(P) + LEN)Q \in A] \tag{4.3}$$

- Case $\delta = 0$: If $R = 0$,

$$\begin{aligned}
&Pr[(M_L(P) + LEN)Q = R] \\
&\qquad = Pr[Q = 0] + Pr[M_L(P) + LEN = 0] \\
&\qquad\qquad - Pr[Q = 0, M_L(P) + LEN = 0]; \\
&\qquad = 2^{-64} + L \times 2^{-64} - L \times 2^{-128}.
\end{aligned}$$

  If $R \neq 0$, then $Q \neq 0$,

$$\begin{aligned}
&Pr[(M_L(P) + LEN)Q = R] \\
&\qquad = Pr[M_L(P) + LEN + Q^{-1}r_0 = 0]; \\
&\qquad = 2^{-64} - L \times 2^{-128}.
\end{aligned}$$

  Thus, $Eqn.(4.3) \leq 2^{-64} + L \times 2^{-64} - L \times 2^{-128} + (2^{32} - 1)(2^{-64} - L \times 2^{-128}) \leq 2^{-32} + L \times 2^{-64}$.

- Case $\delta \neq 0$: By very similar argument, we get

$$Eqn.(4.3) = 2^{-32} - L \times 2^{-96}.$$

Therefore,

$$Eqn.(4.3) \leq \max(2^{-32} + L \times 2^{-64}, 2^{-32} - L \times 2^{-96}).$$

This completes the proof. $\square$

Security under $\mathcal{M}_1$

In this attack model, the adversary can access to the oracle n times and the random numbers P, Q and OTP never change. Then the adversary forges a message-tag pair with the same P, Q and OTP. When $n = 1$, this case is equivalent to $\mathcal{M}_0$. Therefore, we only focus on the case that $n > 1$.

Before the analysis, we present the following lemma.

Lemma 3. *Let Pr_{n_0} and Pr_{n_1} denote the probabilities that the attacker can guess (P, Q, OTP) when he is allowed to query at most n_0 and n_1 times respectively. We have*

$$Pr_{n_0} \leq Pr_{n_1}, \forall n_0 \leq n_1.$$

Proof. If $Pr_{n_0} > Pr_{n_1}$, the attacker can guess based on only n_0 queries. The probability becomes $Pr_{n_0} = Pr_{n_1}$. Therefore, with n_1 queries, the probability is at least Pr_{n_0}. $\square$

As shown by Wu and Gong [111], the substitution forgery probability is one when $n = 2$. By Lemma 3, $\forall n > 1$, the substitution forgery probability is one. Wu and Gong also presented an attacking scenario that the random numbers might repeat.

If attackers can get $n \geq 2$, they can even recover P, Q and OTP. This attack is formally addressed in Theorem 4.

Theorem 4. *Under $\mathcal{M}_1$, the adversary can recover P, Q and OTP with probability*

$$Pr = \begin{cases} \frac{1}{2^{32*(5-n)}}, & 2 \leq n < 5 \\ 1, & n \geq 5 \end{cases}$$

Proof. We prove this theorem by constructing the attack. The attacker first queries the oracle with an empty message to get OTP. Let us assume $OTP = r$. We prove $n = 2$ up to 5 cases. Then the cases that $n > 5$ are straightforward by Lemma 3.

Case n=2: Constructing another message with only one block. Since we know the random mask $OTP = r$, we have

$$Tr_{32}^{64}\left((MP + LEN)Q\right) = a(T + r). \tag{4.4}$$

We consider the case that $T + r = 0$. For $(MP + LEN)Q = 0$, there exist $2 * 2^{64} - 1$ possible (P, Q) pairs. For $(MP + LEN)Q \neq 0$, there are $(2^{32} - 1) * (2^{64} - 1)$ pairs of (P, Q). Thus, $2^{64} * 2^{32} + 2^{64} - 2^{32}$ pairs of (P, Q) satisfy Eqn. (4.4). If $T + r \neq 0$, there are

$2^{64} * 2^{32} - 2^{32}$ pairs of (P, Q) satisfy Eqn. (4.4). Thus, the probability to guess the tuple (P, Q, OTP) is

$$Pr = \frac{1}{2^{32}} \times \frac{1}{2^{96} + 2^{64} - 2^{32}} + \frac{2^{32} - 1}{2^{32}} \times \frac{1}{2^{96} - 2^{32}} \approx \frac{1}{2^{96}}.$$

Case n=3: Construct the second and third messages as

$$\begin{aligned} \overline{\mathbf{M}}_1 &= (b_0, \cdots, b_{k_1-1}), \\ \overline{\mathbf{M}}_2 &= (b_0, \cdots, b_{k_1-1}, \underbrace{0, \cdots, 0}_{k_2-k_1}), \end{aligned}$$

where $b_i \in GF(2)$ for $0 \leq i \leq k_1 - 1$, and $k_1 < k_2 < 32$. After padding, we have

$$\begin{aligned} \overline{\mathbf{M}}'_1 &= (b_0, \cdots, b_{k-1}, \underbrace{0, \cdots, 0}_{64-k_1}), \\ \overline{\mathbf{M}}'_2 &= \overline{\mathbf{M}}'_1. \end{aligned}$$

Let $M'_1 \in GF(q^2)$ denote $(b_0, \cdots, b_{k-1}, \underbrace{0, \cdots, 0}_{64-k_1})$. Note that both messages are one-block messages, and LEN_1, $LEN_2 \in GF(q)$. T_1 and T_2 are computed by

$$\begin{aligned} T_1 &= a^{-1}Tr^{64}_{32}\left((M'_1 P + LEN_1)Q\right) + r, \\ T_2 &= a^{-1}Tr^{64}_{32}\left((M'_1 P + LEN_2)Q\right) + r. \end{aligned}$$

Let $P = p_0 + p_1\alpha$, $M'_1 = m_{10} + m_{11}\alpha$ and $Q = q_0 + q_1\alpha$, where α is defined before; p_i, m_{1i} and $q_i \in GF(q)$, for $i = \{0, 1\}$. Since both k_1 and k_2 are smaller than 32, $m_{10} = 0$. Then,

$$\begin{aligned} & a^{-1}Tr^{64}_{32}\left((M'_1(P) + LEN_1)Q\right) \\ &\qquad = m_{11}\big((q_0 + aq_1)p_0 + (a^2q_1 + bq_1 + aq_0)p_1\big) + LEN_1 q_1, \\ & a^{-1}Tr^{64}_{32}\left((M'_1(P) + LEN_0)Q\right) \\ &\qquad = m_{11}\big((q_0 + aq_1)p_0 + (a^2q_1 + bq_1 + aq_0)p_1\big) + LEN_2 q_1. \end{aligned}$$

Let $(q_0 + aq_1)p_0 + (a^2q_1 + bq_1 + aq_0)p_1\beta = x$ and $q1 = y$. We can solve this linear system. Assume

$$\begin{aligned} q_1 &= s, \\ (q_0 + as)p_0 + (a^2s + bs + aq_0)p_1 &= t. \end{aligned}$$

Guessing $q_0 = e$ with probability $1/2^{32}$, we have

$$(e + as)p_0 + (a^2 s + bs + ae)p_1 = t. \tag{4.5}$$

If $Q = 0$, the number of possible P is 2^{64}. Otherwise, guessing one of p_0 and p_1, we can determine the other one. Therefore, the probability of guessing (P, Q, OTP) is $1/2^{64}$.

Case n=4: Construct another message

$$\overline{\mathbf{M}}_3 = (\underbrace{0, \cdots, 0}_{2^{38}}).$$

Note that the length 2^{38} is the upper bound of the length of messages. The MAC of this message is

$$T_3 = a^{-1} Tr_{32}^{64}(\alpha Q) + r.$$

Since

$$a^{-1} Tr_{32}^{64}(\alpha Q) = q_0 + as,$$

we can get $q_0 = u$. Together with Eqn. (4.5), we have

$$(u + as)p_0 + (a^2 s + bs + au)p_1 = t. \tag{4.6}$$

Guessing either p_0 or p_1 makes this equation become a linear equation of the other one. Therefore, we can solve the unique P. The probability of guessing (P, Q, OTP) is

$$Pr \approx \frac{1}{2^{32}}.$$

Case n=5: Construct another message

$$\overline{\mathbf{M}}_4 = M_1' || (\underbrace{0, \cdots, 0}_{k_4}).$$

The MAC of this message is

$$T_4 = a^{-1} Tr_{32}^{64}\big((M_1' P^2 + LEN_4)Q\big) + r.$$

By the same method above, we can get

$$(u + as)p_0^2 + (bu + a^2 u + a^3 s + a^3 sb)p_1^2 = v.$$

Together with Eqn. (4.6), we have a linear system of p_0^2 and p_1^2.

$$\begin{aligned}(u+as)p_0^2 + (bu + a^2u + a^3s + a^3sb)p_1^2 &= v \\ (u+as)^2p_0^2 + (a^2s + bs + au)^2p_1^2 &= t^2\end{aligned}$$

We can solve unique p_0^2 and p_1^2. Since x^2 is a permutation polynomial over $GF(2^n)$, P is uniquely determined.

By Lemma 3, $\forall n > 5$, (P, Q, OTP) can be uniquely determined. $\square$

Note that in our attack, the opponent first queries an empty message. This query may not be allowed. However, it won't affect our result. Theorem 4 still holds for EIA1. But the proof is more complicated than the one we present here.

Security under $\mathcal{M}_2$

When $n = 1$, this case is equivalent to $\mathcal{M}_0$. We only exam $n > 1$ cases. To prove the security, we present the following lemma.

Lemma 4.

$$Pr[T_i = \beta | T_j = \gamma] = Pr[T_i = \beta], \forall i \neq j.$$

Proof.

$$\begin{aligned}&Pr[T_i = \beta | T_j = \gamma] \\ &= \frac{Pr[T_i = \beta, T_j = \gamma]}{Pr[T_j = \gamma]} \\ &= \frac{\sum_r Pr[T_i = \beta, a^{-1}Tr_{32}^{64}\left((M_j(P_j) + LEN_j)Q\right) = T_j + r]}{\sum_r Pr[a^{-1}Tr_{32}^{64}\left((M_j(P_j) + LEN_j)Q\right) = T_j + r]} \\ &= Pr[T_i = \beta].\end{aligned}$$

$\square$

Lemma 5. *Assume the adversary forges $(\overline{\mathbf{M}}_n, T_n)$ with (P_i, Q_i, OTP_i).*

$$Pr[T_n = \tau_n | T_0 = \tau_0, \cdots, T_{n-1} = \tau_{n-1}] = Pr[T_n = \tau_n | T_i = \tau_i].$$

Proof.

$$\begin{aligned}
&Pr[T_n = \tau_n | T_0 = \tau_0, \cdots, T_{n-1} = \tau_{n-1}] \\
&\quad = \frac{Pr[T_n = \tau_n, T_0 = \tau_0, \cdots, T_{n-1} = \tau_{n-1}]}{Pr[T_0 = \tau_0, \cdots, T_{n-1} = \tau_{n-1}]}
\end{aligned}$$

By Lemma 4, T_i and T_j are independent, $\forall i \neq j$.

$$\begin{aligned}
&Pr[T_n = \tau_n | T_0 = \tau_0, \cdots, T_{n-1} = \tau_{n-1}] \\
&\quad = \frac{Pr[T_n = \tau_n, T_i = \tau_i]}{Pr[T_i = \tau_i]} \\
&\quad = Pr[T_n = \tau_n | T_i = \tau_i].
\end{aligned}$$

□

By Lemma 5, the case that $n > 1$ is equivalent to $\mathcal{M}_0$ as well. Thus, the bound proved before still holds for EIA1 in this case.

Security under $\mathcal{M}_3$

Lemma 6. *Under $\mathcal{M}_3$,*

$$Pr[T_n = \tau_n | T_0 = \tau_0, \cdots, T_{n-1} = \tau_{n-1}] = Pr[T_n = \tau_n].$$

The same argument of Lemma 5 also applies to here. Therefore, we omit the proof. Note that in this case, the attack model is equivalent to impersonation forgery by Stinson's definition, whose success probability is $1/q$.

4.2 An Efficient Implementation of EIA1

This section points out some implementation flaws of the official implementation, and improves the evaluation of polynomial over finite field by an efficient algorithm, which needs only $64 \log r$ multiplications. Compared with the Horner's Rule, which is used in the official EIA1 implementation, this algorithm improves the efficiency dramatically.

4.2.1 The Implementation Flaws of EIA1

The current version of the official implementation is V2.1, which was published in 2009. When we compare this version with the previous one, V1.1, and try to use V2.1 as a comparison baseline, we find several problems.

- **Compiling Error:** There is a compiling error in V2.1. In function $f9$, the last second line uses an undefined symbol $mac32$, which should be $EVAL$.
- **Implementation Bug:** When we compare the output of V1.1 and V2.1, we find that they are totally different. By debugging, we figure out the following implementation errors.
  - The bit arrangements of two versions are different.
  - When a zero-length message is passed in, there is an overflow for both V1.1 and V2.1.

Besides, we find both V1.1 and V2.1 implement the multiplication over $GF(q^2)$ in a quite inefficient way. The function "MUL64xPOW", which can be implemented by a loop, is implemented in a recursive fashion. It is well known that the recursion costs more than the loop because of the manipulation on the stack.

4.2.2 Improvement with Modification to EIA1

If we are allowed to change the algorithm of EIA1, we can improve the efficiency of EIA1 in two ways.

Fix P, Q

For each IK, we fix the random numbers P and Q, which can be done by checking whether the $COUNT$ is equal to 0. When $COUNT = 0$, IK must be a new generated integrity key. In this case, we generate a new pair of (P, Q). Otherwise, we keep using the previous (P, Q). Note that by the arguments of AXU MAC, fixing (P, Q) for each IK does not compromise the security. Originally, the algorithm needs to generate 160 bits to compute the MAC for a message. After the modification, the number of generated bits is reduced to 32 in each round, when $COUNT \neq 0$.

Choose another field

By Lemma 1, if the defining polynomial of $GF(q^2)$ over $GF(q)$ is in the form of $x^2+x+\beta$, the truncation of a multiplication can be computed in such a way

$$\begin{aligned} Truncate(R*Q) &= Tr_{32}^{64}(R*Q) \\ &= r_0q_1 + r_1q_0 + r_1q_1. \end{aligned}$$

In this way, we use only three multiplications over $GF(q)$ to compute the multiplication of $GF(q^2)$ and truncate the product. In the official implementation, the multiplication over $GF(q^2)$ is first computed, and then half of the multiplication result is discarded. Multiplication over $GF(q^2)$ is equivalent to four multiplications over $GF(q)$. The official implementation wastes one multiplication over $GF(q)$.

4.2.3 Improvement without Modification to EIA1

Without changing the algorithm of EIA1, we can improve the efficiency of the algorithm by replacing Horner's Rule with a more efficient polynomial evaluation algorithm and optimizing the multiplication over $GF(q^2)$.

A Faster Polynomial Evaluation Algorithm

The polynomial evaluation algorithm discussed below can be applied to any binary field. Thus it is presented in general binary field $GF(2^n)$, not restrict to $GF(q^2)$. For the purpose of the implementation, field elements are represented under the polynomial basis, which are also used in the official implementation of EIA1.

Let $\mathbb{F}_{2^n}$ be a finite field $GF(2^n)$ and α be a primitive element of $\mathbb{F}_{2^n}$. Then for any $x \in \mathbb{F}_{2^n}$, we can write

$$x = \sum_{i=1}^{n-1} x_i\alpha^i, x_i \in \mathbb{F}_2 \tag{4.7}$$

Let

$$F(x) = \sum_{i=0}^{r} F_i x^i, F_i \in \mathbb{F}_{2^n}.$$

According to (4.7), we can write $F(x)$ as a linear combination of polynomials over $\mathbb{F}_2$ as follows.

$$\begin{aligned} f_i &= \textstyle\sum_{j=0}^{n-1} c_{ij}\alpha_j \Longrightarrow \\ F(x) &= \textstyle\sum_{j=1}^{n-1} \left(\sum_{i=0}^{r} c_{ij}x^i\right)\alpha^j \\ &= \textstyle\sum_{j=1}^{n-1} f_j(x)\alpha^j, \text{where } f_j(x) = \sum_{i=0}^{r} c_{ij}x^i, c_{ij} \in \mathbb{F}_2 \end{aligned}$$

For $P \in \mathbb{F}_{2^n}$, the evaluation of $F(P)$ can be decomposed to computing the evaluation of $f_j(P)$, and the summation of $f_j(P)\alpha^j$. Let $T(r)$ be the number of multiplications used in the evaluation of $f_j(P)$, which is a polynomial over $\mathbb{F}_2$. Since the time complexity is dominated by the number of multiplications, we omit the number of additions and consider the number of multiplications only. Therefore, the complexity of this method for computing $F(P)$, denoted as $\Gamma(r, n)$, is given by

$$\Gamma(r, n) = T(r) \times n.$$

In the following, a fast algorithm to evaluate polynomials over $\mathbb{F}_2$ is presented. Note that for any polynomial $g(x) = \sum_{j=0}^{r} g_jx^j, g_j \in \mathbb{F}_2$, it can be decomposed into a summation of two polynomials by collecting odd and even exponents of x as follows:

$$g(x) = g_0(x^2) + xg_1(x^2) = g_0(x)^2 + xg_1(x)^2$$

where both $g_i(x)$'s have their respective degrees at most of $r/2$. Algorithm 2 presents the details of the above method [51].

Algorithm 2: Polynomial evaluation

```
Data: Input: $g(x) \in \mathbb{F}_2[x]$ and $S = \{P^i : i = 2, \cdots, r/2^k\}$
Result: Output: $g(P)$
if $deg\,(g(x)) \leq r/2^k$ then
    return evaluation of $g(x)$ using $S$;
else
    $g(x) = g_0(x)^2 + xg_1(x)^2$;
    recursively call this procedure on $g_0(x)$ and $g_1(x)$;
    return evaluation of $g_0(x)^2 + xg_1(x)^2$;
end
```

Optimize the Code

The recursive function call in "MUL64XPOW" function is replaced by a loop, and all the integer multiplications of 2^n is implemented by shifting n bits to the left. The efficiency improvement is analyzed in the next section.

4.2.4 Efficiency Analysis

Three versions of EIA1 are compared in this research: the first one that evaluates the polynomial using Algorithm 2, the second one that implements the multiplication of $\mathbb{F}_{2^n}$ in loops instead of recursive function call, and the official implementation. Denote the first, second, and the official implementations as $f9_{eva}$, $f9_{opt}$, and $f9$, respectively.

The efficiency of $f9_{eva}$

The complexity of Algorithm 2 consists of

- At most $r/2^k$ multiplications for computing S.
- At most r additions (at most $r/2^k$ additions to evaluate each polynomial at the k-th step), which can be reduced to $r/\ln r$ by reusing some terms (dynamic programming).
- $2^k - 1$ multiplications to get the final result by substituting back the values.

For the same reason mentioned above, we consider the number of multiplications only. Thus, the complexity of evaluating a polynomial over $\mathbb{F}_2$ is upper bounded by $T(r) \le r/2^k + 2^k$ multiplications in $\mathbb{F}_{2^n}$.

Selection of k: Purpose of selection of k is to minimize $T(r) \le r/2^k + 2^k$. Note that

$$\frac{dT(r)}{dk} = 2^k ln2 - \frac{rln2}{2^k}.$$

So when we take

$$2^k = \frac{r}{2^k} \Longrightarrow k = \frac{1}{2}\log r, \tag{4.8}$$
$$T(r) \le \log r.$$

Squaring: For the squaring, one way is pre-computing the list and storing that

$$\alpha^{2i}, i = 0, 1, \cdots, n-1.$$

In the EIA1, $n = 64$ and α is a root of the polynomial which defines $\mathbb{F}_{2^{64}}$. Another way is computing the squaring under the normal basis. By exhaustive search, we find one normal element $0\text{x}C48F6B490773E7A1$.

Therefore, the total complexity of evaluating $F(P)$ is given by $n \log r$. Compared with the Horner's Rule, this number is smaller when the degree of the polynomial is greater than a threshold. For $n = 64$, the threshold is 589.

The efficiency of $f9_{opt}$

In this version, the optimization includes the follows.

- Replacing all the integer multiplications of 2^n by shifting n bits to the left;
- Changing the recursive implementation of the multiplication of $\mathbb{F}_{2^n}$ to the loop implementation.

Table 4.1 lists the difference between multiplication of 2^n and shifting n bits. The time consumption is measured when the multiplication or shift is computed 2^{30} times. Although the difference is quite tiny, it improves the efficiency a little bit.

Table 4.1: Time costs of multiplying by 2^n and shifting n bits

n	Multiplication (clock cycle)	Shift (clock cycle)
1	7531852.000000	7497074.500000
2	7541375.000000	7537943.000000
3	7887745.500000	7832369.000000

The efficiency difference between the recursion and loop is list in Table 4.2. This improvement is much more significant than replacing multiplication of 2^n with shifting n bits.

The comparison of our implementation, our optimization and the official implementation is shown in Figure 7.1. We call our implementation EIA1, the optimized implementation $f9_{opt}$ and the official implementation $f9$.

Table 4.2: Time costs of Recursion and Loop

	Recursion (clock cycle)	Loop (clock cycle)
Time consumption	13.642400	5.600540

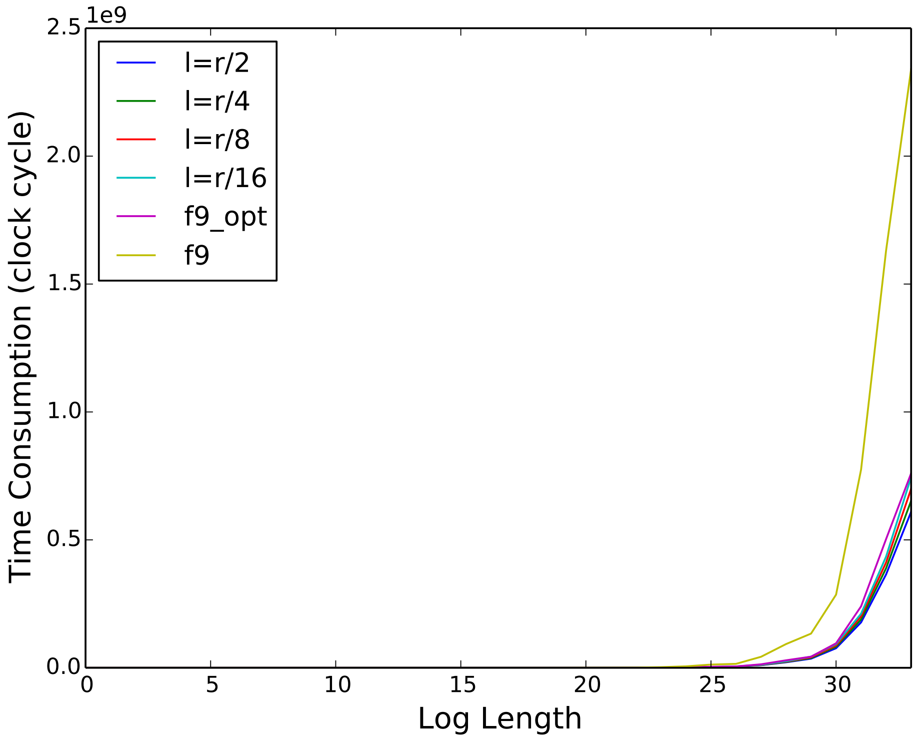


Figure 4.1: Comparison of $f9$, $f9_{opt}$ and $f9_{eva}$

In Figure 7.1, the curve $l = r/2^i$ shows the time consumption when $k = i$. Theoretically, as long as $k < \log r$, the larger k is, the higher speed $f9_{eva}$ should achieve. However, Figure 7.1 suggests opposite results. The reason is that allocating memories costs much more than computing multiplications. When the length of a message is large enough, $f9_{eva}$ is always better than $f9$ and $f9_{opt}$. Theoretically, this threshold should be 589, but because of the memory allocation, the real threshold is greater than the theoretical one.

4.3 Summary

In this chapter, we prove that the conditional probability and the unconditional probability in Krawczyk's proof are equal. The equality fixes Krawczyk's proof, which states that AXU MACs are secure. Then, the security of EIA1 is assessed under different models. The assessment shows that only if the random numbers are all repeated, EIA1 has some security issues. Specifically, if the random numbers are reused, the linear forgery attack can be directly applied to EIA1. In addition, if the random numbers are allowed to reuse more than twice, attackers have non negligible probability to recover the random numbers. Especially, when the random numbers are rescued for more than four times, attackers can recover the random numbers determinately.

After the theoretical analysis, we examine the implementation provided in the proposal of EIA1, and find several implementation flaws. The implementation in the proposal is very inefficient. We optimize their codes and get an optimized version called $f9_{opt}$. Moreover, we replace the Horner's Rule with our efficient polynomial evaluation algorithm to get a new implementation called $f9_{eva}$. By the comparison among $f9$, $f9_{opt}$ and $f9_{eva}$, we can conclude that our optimized version works faster than the original one, and our polynomial evaluation method is faster than Horner's Rule when the length of the message is large enough.

Chapter 5

Security Proof of TUAK

In addition to the EIA family, many other MACs exist in the 4G LTE system. For example, f_1, f_2 functions introduced in Section 1 are two MACs in the AKA procedure. Originally MILENAGE is the only cipher suite implementing A3/A8. After SHA-3 was standardized, Vodafone proposed another cipher suite called TUAK, which also followed A3/A8 specification, and was built upon Keccak. This chapter presents the security proof of TUAK as message authentication codes and key derivation functions. The proofs prove the security of TUAK using certain properties of Keccak.

The rest of this chapter is organized as follows. Section 5.1 proves the security of f_1, f_1^* and f_2 functions as MACs. Section 5.2 proves other f_n functions as KDFs. The last section concludes this chapter.

5.1 Security proof of the f_1, f_1^* and f_2 function construction

The f_1 and f_1^* functions are two MAC algorithms. The f_2 function is a special MAC algorithm, because it generates the response, which is essentially a MAC, in the authentication process. The threat model of f_2 is the same as MACs' threat model. Thus, f_2 should resist to all the attacks applied to MACs. This section investigates the properties of these three functions as MAC algorithms. At the beginning, f_1 is taken as an example to derive the security bound. After that, this security bound is generalized to f_1^* and f_2.

By the definition of the f_1 function, we may consider the first 768 bits (TOP_c, $INSTANCE$, $ALGONAME$, $RAND$, AMF, SQN, K) as the domain separator, message and key, the

320 bits in the middle as the padding, and the last 512 zeros as the capacity. Specifically, the bits $\underbrace{1\cdots1}_{5}\underbrace{0\cdots0}_{314}1$ are the *Sakura* padding [26]. Let f_1' be same the function as f_1 but the first 768 bits could be any value in $\mathbb{F}_{2^{768}}$, i.e.

$$f_1(x) = f_1'(TOP_c||INSTANCE||ALGONAME||x||K).$$

Fact 1. *The f_1' function is a sponge function with the bit rate $r = 1088$ and the capacity $c = 512$. The f_1 function is f_1' in MAC mode.*

In the following discussion, we always assume the underlying permutation Π is a pseudorandom permutation (PRP).

Lemma 7. *Throw u balls into v buckets uniformly, where $u \geq v$. Let event $\mathcal{E}$ denote the event that at least one bucket is empty. The probability that event $\mathcal{E}$ happens is upper bounded by*

$$Pr[\mathcal{E}] < v(1 - v^{-1})^u. \tag{5.1}$$

Proof. after throwing all u balls, denote the event that the i-th bucket is empty as $\mathcal{E}_i$. The probability that event $\mathcal{E}_i$ happens is given by

$$Pr[\mathcal{E}_i] = (1 - v^{-1})^u.$$

Since $\mathcal{E}_i$ and $\mathcal{E}_j$ are not independent for $i \neq j$, and $u \geq v$, we have

$$Pr[\mathcal{E}] < \sum_{i=0}^{v-1} \mathcal{E}_i = v(1 - v^{-1})^u.$$

This completes the proof □

Lemma 8. *Throw u balls into v buckets uniformly, where $u < v$. Let event $\mathcal{E}$ denote the event that at least $v - u + 1$ bucket is empty. The probability that event $\mathcal{E}$ happens is upper bounded by*

$$Pr[\mathcal{E}] < \begin{pmatrix} v \\ u \end{pmatrix} u(1 - u^{-1})^u.$$

Proof. Let $\mathcal{I}$ be an index set of the bucket, $\mathcal{I} = \{i|0 \leq i \leq v - 1\}$. Each time, we select a subset of $\mathcal{I}$, and the size of this subset is $v - u$. We call this subset $\mathcal{I}_j$. Let the buckets indexed by the subset $\mathcal{I}_j$ be empty. Then, the problem becomes Lemma 7. Let the event $\mathcal{E}_j$ denote fixing subset $\mathcal{I}_j$, the rest buckets have at least one empty bucket. By Lemma 7,

$$Pr[\mathcal{E}_j] < u(1-u^{-1})^u.$$

By the same argument as the proof of Lemma 7,

$$Pr[\mathcal{E}] < \sum_{j=0}^{w} u(1-u^{-1})^u, \text{ where } w = \left(\begin{array}{c} \text{v} \\ \text{v-u} \end{array}\right) - 1.$$

Therefore,

$$Pr[\mathcal{E}] < \left(\begin{array}{c} \text{v} \\ \text{u} \end{array}\right) u(1-u^{-1})^u.$$

This proves the Lemma. $\square$

Lemma 9. *The output of f_1' is uniformly distributed, which means $\forall y \in \mathbb{F}_2^n$, randomly chosen an $x \in \mathbb{F}_2^{768}$, the following equation always holds for f_1'.*

$$Pr[f_1'(x) = y] = \frac{1}{2^n} \pm \epsilon,$$

where $n (= 32, 64, 128, 256)$ is the length of the output, and ϵ is a negligible value.

Proof. Notice the underlying block Π is a pseudorandom permutation. Thus, the truncated output is uniformly distributed. Now, we restrict the input of Π to a subspace $\mathbb{F}_2^{768}$ by setting 832 bits to a constant to get f_1'. Let Ω and Γ denote the pre-image and image set of f_1' respectively. In this case $|\Omega| = 2^{768}$. By Lemma 7, $|\Gamma| = 2^n$ with overwhelming probability. Since the truncated output of Π is uniformly distributed, each value in Γ has almost the same number of pre-image. Then

$$Pr[f_1'(x) = y] = \frac{1}{2^n} \pm \epsilon.$$

$\square$

Theorem 5. *Randomly pick x_0 and x_1 from the pre-image set of f_1. The probability of finding a collision of f_1 is given by*

$$Pr[f_1(x_0) = f_1(x_1)] = \begin{cases} \frac{1}{2^n} \pm \epsilon & \textit{if } n = 32, 64, 128, \\ \epsilon & \textit{if } n = 256. \end{cases}$$

Proof. First, let us consider the case that $n = 32, 64, 128$. By the proof of Lemma 9, the size of image set of f_1 is also 2^n with the probability approaching 1. By Lemma 9, the outputs of f_1' distribute in $\mathbb{F}_2^n$ uniformly. Thus, the outputs of f_1 are distributed uniformly in $\mathbb{F}_2^n$. Then we have

$$Pr[f_1(x_0) = f_1(x_1)] = \sum_{y \in \mathbb{F}_2^n} Pr[f_1(x_0) = y, f_1(x_1) = y].$$

Since Π is assumed to be a PRP, $f(x_0)$ and $f(x_1)$ are independent for $x_0 \neq x_1$. Therefore,

$$Pr[f_1(x_0) = f_1(x_1)] = \frac{1}{2^n} \pm \epsilon.$$

Then, let us consider the case that $n = 256$. Notice that $|\Omega| = 2^{192}$. It is far less than 2^n. By Lemma 8, $|\Gamma| = |\Omega|$. The probability of collision is ϵ.

This completes the proof. □

Before the next theorem, we clarify some notations. Assume there is an oracle $\mathcal{O}_a$, which can recover the key of f_1 with N queries. Using $\mathcal{O}_a$, attackers can build a game $\mathcal{G}$ to find the pre-image of f_1'. We denote l as the number of queries that attackers need to find the pre-image of f_1' using any method except the game $\mathcal{G}$. The notation b is the lower bound of the overall expected number of queries to find the pre-image of f_1'.

Theorem 6. *The expected success rate of the universal key recovery attack with N queries under the adaptive chosen plaintext attack on f_1 function is no greater than*

$$\frac{2^m(2^k + (1 - \frac{|\Gamma|}{2^m})l - b)}{|\Gamma|(2^k - N)},$$

where m is the length of the output; k is the size of the key; Γ is the image set of f_1.

Proof. Let us assume there exist two oracles, $\mathcal{O}_t^K$ and $\mathcal{O}_a$. $\mathcal{O}_t^K$ returns the output of function f_1 (denoted by MAC) with the key K given an triple of $(RAND, SQN, AMF)$. $\mathcal{O}_a$ chooses N triples of $(RAND, SQN, AMF)$ adaptively and queries $\mathcal{O}_t^K$ for the MAC of each triple. After that, the oracle $\mathcal{O}_a$ returns the key K used by $\mathcal{O}_t^K$. The behavior is described in Table 5.1.

Notice that $\mathcal{O}_a$ is an universal key recovery attack under the adaptive chosen plaintext model on f_1. Using $\mathcal{O}_t^K$ and $\mathcal{O}_a$, one attacker can construct a probabilistic game $\mathcal{G}$ to find

Table 5.1: $\mathcal{O}_t^K$ and $\mathcal{O}_a$.

$\mathcal{O}_t^K$	$\mathcal{O}_a$
Input: $T = (RAND, SQN, AMF)$	Input: no input
Output: MAC	Output: K or failed
Compute $MAC = f_1(K, T)$; Return MAC.	Choose $T_0 = (RAND_0, SQN_0, AMF_0)$; query $\mathcal{O}_t^K$ with T_0; and get H_0; $\cdots$ Choose $T_{N-1} = (RAND_{N-1}, SQN_{N-1}, AMF_{N-1})$ based on the previous selection of $T_0 \cdots T_{N-2}$ and the output; query $\mathcal{O}_t^K$ with T_{N-1}; and get H_{N-1}; Compute $K = A(T_0, \cdots, T_{N-1}, H_0, \cdots H_{N-1})$ with success probability p; Return K or failed.

Table 5.2: The game $\mathcal{G}$ to find the pre-image of f_1'.

$\mathcal{G}$
Input: $H = f_1'(x)$, x is not a pre-image of any f_1.
Output: The state of f_1 s.t. $H = f_1(T_s)$ or failed.
1. If $H \notin \Gamma$, return failed; 2. If $H = H_s$, query $\mathcal{O}_a$ to recover K; 3. Construct the state by T_s, K, and return the state.

the pre-image of one output of f_1'. Let $T_i = (RAND_i, SQN_i, AMF_i)$ $(0 \le i \le 2^{192} - 1)$ denote all possible input of $\mathcal{O}_t^K$, then set Γ is defined by $\Gamma = \{H_i : H_i = \mathcal{O}_t^K(T_i), 0 \le i \le 2^{192} - 1\}$. The game is shown in Table 5.2.

Notice that the condition of this adversary is choosing a $H = f_1'(x)$ and x is not a pre-image of any f_1. The cardinality of the pre-image set of f_1 is 2^{192}, while the cardinality of the pre-image set of f_1' is 2^{768}. The probability that given an x, x is a pre-image of f_1 is $1/2^{576}$. This probability is negligible. Thus, in the following discussion, we only consider x is not a pre-image of any f_1.

Since we assume the output of the f_1' function is uniformly distributed, the probability that given any $H \in \mathbb{F}_2^m$ $H \in \Gamma$ is given by

$$Pr[H \in \Gamma] = \frac{|\Gamma|}{2^m},$$

where m is the length of the output of f_1. In the following, we discuss two cases, $H \in \Gamma$ and $H \notin \Gamma$.

If $H \in \Gamma$, the attack may use game $\mathcal{G}$ to find the pre-image. Assume the probability that $\mathcal{G}$ can recover the key is

$$Pr[\mathcal{G}\ success] = p.$$

However, game $\mathcal{G}$ may fail with probability $1-p$. Then, the attacker needs 2^k queries to recover the key.

If $H \notin \Gamma$, the attacker needs at most 2^k queries to let the game $\mathcal{G}$ output failed, and l queries to find the pre-image by other methods. Thus, the expected number of queries to find the pre-image of one output is given by

$$\frac{|\Gamma|}{2^m}p(N-2^k)+\frac{|\Gamma|}{2^m}2^k+(1-\frac{|\Gamma|}{2^m})(2^k+l).$$

Since b is the lower bound of the expected number, thus

$$\frac{|\Gamma|}{2^m}p(N-2^k)+\frac{|\Gamma|}{2^m}2^k+(1-\frac{|\Gamma|}{2^m})(2^k+l)\geq b.$$

Therefore, we have

$$p \leq \frac{2^m(2^k+(1-\frac{|\Gamma|}{2^m})l-b)}{|\Gamma|(2^k-N)},$$

This completes the proof. □

Because $|\Gamma|/2^m \approx 1$, the result of Theorem 6 becomes

$$p<\frac{2^k-b}{2^k-N}.$$

By Bertoni *et al.* [60], the value of b is bounded by $2^{n-r}+2^{c/2}$. Therefore, as long as N is far smaller than b, no significant universal key recovery attack exists. Formally, we have the following corollary.

Corollary 2. *When $N << b$, the probability of universal key recovery attack is negligible.*

Substitution forgery attacks on TUAK is described as follows. Attackers are allowed to select N messages adaptively and to access the oracle $\mathcal{O}_t^K$ to get the MAC of each query. After N queries, attackers try to forge the MAC of a message other than the previously queried messages.

Theorem 7. *Under the model described above, the expected number of queries to successfully forge a MAC generated by f_1 is given by*

$$Exp(N) = 2^{320}.$$

Proof. In the following proof, we will use K to denote the key, c to denote the capacity, and z to denote the maximum number of messages that can be MACed with the same key. By Bertoni *et al.* [60], when using a sponge function as a MAC algorithm, the workload to forge a tag generated by this MAC algorithm is given by

$$Exp(N) = \frac{2^c}{z-1},$$

when $|K| < c - \log_2(z)$.

For f_1 case, $z = 2^{192}$, $|K| = 256(or\ 128)$, $c = 512$. Then we have

$$Exp(N) = \frac{2^{512}}{2^{192}-1} \approx 2^{320}.$$

□

Corollary 3. *It is infeasible to forge a MAC generated by f_1 under the model described above.*

The workload in Theorem 7 is greater than 2^{256}, which is the workload of guessing the key. Thus, f_1 resists to substitution forgery attacks. For functions f_1^* and f_2, we have the following arguments.

- f_1 and f_1^* are almost the same. The only difference is the constant, but that does not affect the properties of the function. Therefore, all result applied to f_1 can also be applied to f_1^*. We omit the arguments of f_1^*.
- The f_2 function has different pre-image set compared with f_1 and f_1^*. SQN and AMF does not exist in the input of f_2. The cardinality of the pre-image set becomes 2^{128}.

Theorem 6 and its corollary also holds for f_2. Theorem 7 needs some tiny modification.

Theorem 8. *The expected number of queries to successfully forge a MAC generated by f_2 is*

$$Exp(N) = 2^{384}.$$

From the proof of Theorem 7, and the cardinality of the pre-image set of f_2, Theorem 8 is straightforward. Corollary 3 still holds for f_2 because this workload is also greater than guessing the key.

5.2 Security proof of f_3, f_4, f_5 and f_5^* construction

Functions f_3, f_4, and f_5 are KDFs. The basic requirements of KDFs are: 1) given the random number, adversaries should not have the ability to guess the derived key; 2) for two different random numbers, each of the f_3 - f_5 functions should not produce the same derived key; 3) KDFs must protect long term credentials from being exposed when the adversaries obtain certain number of derived keys.

Remark 2. *Varying the constant, INSTANCE, makes the same underlying permutation, Π, work as different functions. However, the f_2 - f_5 functions share the same INSTANCE. Therefore, the outputs of f_2 - f_5 functions are different portions of the output generated by the same function. The sponge function always requires the output length smaller then half of the capacity. In this case, the total length of the output generated by the same INSTANCE lies between 336 and 816. Even we consider the effective capacity of TUAK is 768, the output length 816 is still far greater than half of 768. Although we cannot find any immediate attacks, this construction has potential risks.*

Using different $INSTANCE$ values for f_2 - f_5 can improve the security of TUAK, but it compromises efficiency. As long as the speed of TUAK fulfills the specification of 3GPP, security is more important than efficiency. Therefore, the following results are all based on the revised version of f_2 - f_5.

Since the revised version of f_3 - f_5 are the same as f_2, the arguments on f_2 in the last section can be applied to f_3 - f_5. Therefore, we only list the theorems below without any proof. To clearly illustrate following results, define f_i' for $3 \leq i \leq 5$ in the same way as f_1'.

Theorem 9. *Randomly pick x_0 and x_1 from the pre-image set of f_i ($3 \leq i \leq 5$). The probability of finding a collision of f_i is given by*

$$Pr[f_i(x_0) = f_i(x_1)] = \begin{cases} \frac{1}{2^n} \pm \epsilon & \text{if } n = 32, 64, 128, \\ \epsilon & \text{if } n = 256. \end{cases}$$

Theorem 9 implies that the probability that f_i ($3 \leq i \leq 5$) generates the same derived key for different inputs is negligible.

Theorem 10. *Under the CPA model, the expected success rate of universal key recovery attacks on f_i ($3 \leq i \leq 5$) function with N queries is no greater than*

$$\frac{2^m(2^k + (1 - \frac{|\Gamma|}{2^m})l - b)}{|\Gamma|(2^k - N)},$$

where m is the length of the output; k is the size of the key; Γ is the image set of f_i.

This theorem means that opponents can hardly recover long term credentials.

Theorem 11. *The expected number of queries to successfully forge a derived key generated by f_i ($3 \leq i \leq 5$) is*

$$Exp(N) = 2^{384}.$$

Theorem 11 shows that forgery of a derived key generated by f_i ($3 \leq i \leq 5$) without knowing the long term key is impossible. The f_5^* function is the same as the revised f_5, except the constant, INSTANCE. The above theorems also hold for f_5^*.

5.3 Summary

In this chapter, we proved the security of f_i ($1 \leq i \leq 5$) and f_i^* ($i = 1, 5$) as MAC algorithms and key derivation functions. For the MAC functions, we consider the complexity of universally forging a MAC, recovering the key, and finding a collision. For the KDF functions, we consider the complexity of recovering the master key, guessing the derived key, and finding a collision for the derived keys.

When analyzing TUAK, we found that the constructions of f_2, f_3, f_4, and f_5 are not in a recommended way. Since they all use the same constant as the input, these constructions output more bits than it is allowed by the security capacity of Keccak. We fixed the constructions by varying the constants for each function to meet the requirement of Keccak. The security of these functions are proved on the revised version.

The functions f_1, f_2 and f_1^* as three MACs are secure in the sense of key recovery, and substitution forgery. We prove that as long as the underlying primitive, Keccack, resists pre-image attacks, recovering the key of f_1, f_2, and f_1^* is infeasible. Based on some properties of Keccak, we show that finding two messages with the same MAC, or given one MAC forging another are both infeasible as well.

The functions f_3, f_4, and f_5 are KDFs. Using the properties of Keccak, we prove these three KDFs are secure, which means recovering the master key, guessing the derived keys, and finding to identical derived keys are all infeasible.

Chapter 6

Multi-Output Filtering Model

When assessing the security of TUAK, we developed a useful tool called Multi-Output Filtering Model (MOFM) to study the randomness property of a cryptographic primitive, such as message authentication codes or block ciphers. The MOFM consists of an LFSR and a multi-output filtering function. The content in this Chapter is twofold. First, we propose an attack technique under IND-CPA using the MOFM. By introducing a distinguishing function, we theoretically determine the success rate of this attack. In particular, we construct a distinguishing function based on the distribution of the linear complexity of component sequences, and apply it on studying TUAK's f_1 algorithm, AES, KASUMI and PRESENT. We demonstrate that the success rate of the attack on KASUMI and PRESENT is non-negligible, but f_1 and AES are resistant to this attack. Second, we study the distribution of the cryptographic properties of component functions of a random primitive in the MOFM. Our experiments show some non-randomness in the distribution of algebraic degree and nonlinearity for KASUMI.

The rest of this Chapter is organized as follows. Section 6.1 introduces the preliminaries only for this section only. In Section 6.2, we first describe the MOFM in which an LFSR is used to generate the inputs to a multi-output function. In Section 6.3, we describe the attack model of our distinguishing attack. Section 6.4 presents the construction of a distinguishing function based on the distribution of linear complexity of component sequences. In Section 6.5, we present some non-randomness in the distribution of the algebraic degree and the nonlinearity of component functions of f_1 and other primitives. Section 6.6 concludes this Chapter.

6.1 Basic Definitions

We present some definitions, which will be used in this Chapter.

6.1.1 Basic definitions on sequences

We present some definitions on sequences. For a well-rounded treatment of sequences and Boolean functions, the reader is referred to [33, 57].

Let $\mathbf{s} = \{s_i\}$ be a sequence generated by an LFSR whose recurrence relation is defined as

$$s_{\ell+i} = \sum_{j=0}^{\ell-1} c_j s_{i+j}, s_i, c_i \in \mathbb{F}_2,\ i = 0, 1, ... \tag{6.1}$$

where $p(x) = \sum_{i=1}^{\ell} c_i x^i \in \mathbb{F}_2[x]$ is the characteristic polynomial of degree ℓ of the LFSR. A binary sequence $\mathbf{s}$ in Eq. (6.1) with period $2^\ell - 1$ generated by an LFSR is called an *m-sequence*. Let $\mathbf{s} = \{s_i\}$ be an m-sequence of period $2^\ell - 1$ and $f(x_0, ..., x_{\ell-1})$ be a Boolean function in ℓ variables. We define a sequence $\mathbf{a} = \{a_i\}$ as

$$a_i = f(s_{r_1+i}, s_{r_2+i}, ..., s_{r_t+i}),\ s_i, a_i \in \mathbb{F}_2,\ i \geq 0$$

where $r_1 < r_2 < \ldots < r_t < \ell$ are tap positions. Then the sequence $\mathbf{a}$ is called a *filtering sequence* and the period of $\mathbf{a}$ equals $2^\ell - 1$.

The *linear complexity* or *linear span* of a sequence is defined as the length of the shortest LFSR that generates the sequence. For an m-sequence, the linear complexity of an m-sequence is equal to the length of its LFSR [57]. On the other hand, the linear complexity of a nonlinear filtering sequence lies in the range of ℓ and $2^\ell - 1$ [72]. If a filtering sequence has linear complexity $2^\ell - 1$, then we call it has *optimal* linear complexity.

6.1.2 Basic definitions on Boolean functions

There is a one-to-one correspondence between a sequence and a Boolean function. The correspondence between a Boolean function and a sequence can be obtained by computing the trace representation of a given sequence using the Fourier transformations. For the details, see Chapter 6 of [57].

Definition 11. *Let f be a Boolean function from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. Then f can be uniquely represented by its algebraic normal form (ANF) as*

$$f(x) = \sum_{I \in \mathcal{P}(\{0,\ldots,n-1\})} a_I x^I,$$

where $a_I \in \mathbb{F}_2, x^I = \prod_{i \in I} x_i$ and $\mathcal{P}(\{0,\ldots,n-1\})$ is the power set of $\{0,\ldots,n-1\}$. The algebraic degree of f, denoted by $d(f)$, is the maximal size of I in the ANF of f such that $a_I \neq 0$.

One of the most important properties of Boolean functions is its nonlinearity, which was proposed to measure the distance of it to all affine functions. A cryptographic strong Boolean function is supposed to have high nonlinearity to resist linear attacks [81].

Definition 12. *The Walsh spectrum of a Boolean function f to a point $a \in \mathbb{F}_2^n$, denoted by $W_f(a)$, is defined by*

$$W_f(a) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x)+a \cdot x}$$

where $a \cdot x$ is the inner product of a and x.

The *nonlinearity* of f can be defined in terms of the Walsh spectrum as

$$\mathrm{NL}(f) = 2^{n-1} - \max_{a \in \mathbb{F}_2^n} \frac{|W_f(a)|}{2}.$$

When n is an even positive integer, it is known that the maximum value if the nonlinearity of a Boolean function f is $\mathrm{NL}(f) \geq 2^{n-1} - 2^{n/2-1}$ [33]. A Boolean functions achieving this bound is called a *bent function*.

Let m and n be two positive integers. A function F, from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$, defined by $F(x) = (f_1(x), f_2(x), ..., f_m(x))$ is called an (n,m)-function, a multi-output Boolean function, or a vectorial Boolean function, where f_i's are called coordinate functions [33].

6.2 Multi-Output Filtering Model

In this section, we provide a detailed description of the MOFM of a cryptographic primitive.

6.2.1 Description of the Multi-Output Filtering Model

Let $\mathbf{a} = \{a_i\}_{i\geq 0}$ be a binary sequence generated by an ℓ-stage LFSR whose recurrence relation is

$$a_{\ell+i} = \sum_{j=0}^{\ell-1} c_j a_{i+j},\ c_j \in \mathbb{F}_2,\ i \geq 0, \tag{6.2}$$

where $p(x) = x^\ell + \sum_{i=0}^{\ell-1} c_i x^i$ is a primitive polynomial of degree ℓ over $\mathbb{F}_2$ and $\mathrm{STATE}_j = (a_j, a_{j+1}, ..., a_{\ell-1+j})$ is called the j-th state of the LFSR. Using this LFSR, from the above sequence $\mathbf{a}$, we generate a set of messages of n bits, denoted by $\mathcal{R} = \{\mathrm{R}_j : 0 \leq j \leq 2^\ell - 2\}$ where

$$\mathrm{R}_j = (a_j, a_{j+1}, \cdots, a_{j+n-1}),\ j = 0, 1, ..., 2^\ell - 2, \tag{6.3}$$

Here modulo $2^\ell - 1$ is taken over the indices of a_i's. Note that the elements in $\mathcal{R}$ are in the sequential order. We now define the MOFM on $F : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^m$. For a fixed key K and for each R_j with $0 \leq j \leq 2^\ell - 2$, we obtain

$$\begin{aligned} \mathrm{C}_j &= F(\mathrm{K}, \mathrm{R}_j) \\ &= (g_0(\mathrm{K}, \mathrm{R}_j), \ldots, g_{m-1}(\mathrm{K}, \mathrm{R}_j)) \\ &\triangleq (y_{j,0}, y_{j,1}, \ldots, y_{j,m-1}). \end{aligned} \tag{6.4}$$

Using a matrix, we can represent the above C_j as

$$\begin{pmatrix} \mathrm{C}_0 \\ \mathrm{C}_1 \\ \vdots \\ \mathrm{C}_{2^\ell-2} \end{pmatrix} = \begin{pmatrix} y_{0,0} & y_{0,1} & \cdots & y_{0,m-1} \\ y_{1,0} & y_{1,1} & \cdots & y_{1,m-1} \\ \vdots & \vdots & & \vdots \\ y_{2^\ell-2,0} & y_{2^\ell-2,1} & \cdots & y_{2^\ell-2,m-1} \end{pmatrix}. \tag{6.5}$$

The matrix (6.5) provides us two methods to study cryptographic properties of F as described below.

I. **Sequence point of view:** Each column in the above can be considered as a sequence of period $2^\ell - 1$ for a nonzero initial state of the LFSR. Each sequence of period $2^\ell - 1$ is called a *component sequence*. We denote the i-th component sequence by $\mathbf{s}_i$ and $\mathbf{s}_i = \{y_{0,i}, y_{1,i}, ..., y_{2^\ell-2,i}\}$. $\mathbf{s}_i$ can also be considered as a filtering sequence with filter function g_i, $0 \leq i \leq m-1$.

II. **Boolean function point of view:** From (6.4) and (6.5), we see the following process

$$g_i : \left\{\mathrm{STATE}_j \in \mathbb{F}_2^\ell \text{ of the LFSR}\right\} \rightarrow \{\mathrm{R}_j \in \mathbb{F}_2^n\} \rightarrow i\text{-th component sequence.}$$

Therefore, each component sequence can also be regarded as a Boolean function on $\mathbb{F}_2^\ell$. Note that, for a nonzero initial state, the LFSR cannot generate all-zero state, we need to query F to get the output value $F(\mathrm{K}, 0^n)$ for all-zero input for all component Boolean functions. With a fixed K in F, using an ℓ-stage LFSR, we obtain m Boolean functions on $\mathbb{F}_2^\ell$. Mathematically, m Boolean functions $g_i : \mathbb{F}_{2^\ell} \rightarrow \mathbb{F}_2$ $(0 \leq i \leq m-1)$ are defined as

$$g_i(K, \mathrm{STATE}_j) = y_{j,i}, \quad (0 \leq j \leq 2^\ell - 2). \tag{6.6}$$

We call each Boolean function g_i a *component* or *coordinate function* of F.

6.2.2 Application to TUAK's f_1, AES, KASUMI and PREENT

For the sake of clarity on the input assignment, we briefly explain how we apply the MOFM on TUAK's f_1, and block ciphers AES, PRESENT and KASUMI.

TUAK's f_1:

Recall that f_1 takes K, RAND, and SQN as inputs. Now we fix a key K and a sequence number SQN. We use an ℓ-stage LFSR to generate random numbers RAND_j in f_1. Denoting by the i-th state of the ℓ-stage LFSR by $\mathrm{STATE}_i \in \mathbb{F}_2^\ell$. We obtain $2^\ell - 1$ different n-bit RAND numbers $\mathcal{R} = \{\mathrm{R}_j : 0 \leq j \leq 2^\ell - 2\}$ by Eq. (6.3) and the component sequences and component functions are obtained using Eq. (6.4) with $\mathrm{C}_j = f_1(\mathrm{K}, \mathrm{R}_j, \mathrm{SQN})$.

Remark 3. *For TUAK's f_1 function, in Eq. (6.5), recovering the last bit $y_{2^\ell-2,i}$ for each component sequence $\boldsymbol{s}_i$ from the previous $2^\ell - 2$ bits is equivalent to recovering $C_{2^\ell-2}$ from $\{C_0, ..., C_{2^\ell-3}\}$. This leads to a MAC forgery attack on f_1.*

AES, PRESENT and KASUMI:

Recall that AES-128 accepts a 128-bit key and a 128-bit input and produces an output of 128 bits, and AES-256 accepts a 256-bit key and a 128-bit input and produces an output of 128 bits [40]. KASUMI has a 64-bit input, a 128-bit key, and a 64-bit output. For AES-128 and AES-256, the inputs messages of 128 bits are generated using an LFSR of length ℓ and by Eq. (6.3), and the component sequences and functions are obtained using Eq. (6.4) with $\mathrm{C}_j = \text{AES-128}\,(\mathrm{K}, \mathrm{R}_j)$ and $\mathrm{C}_j = \text{AES-256}\,(\mathrm{K}, \mathrm{R}_j)$. PRESENT [29] is a 64-bit block cipher with a 80-bit key. The component sequences and functions of PRESENT are obtained using Eq. (6.4) with $\mathrm{C}_j = \mathrm{PRESENT}\,(\mathrm{K}, \mathrm{R}_j)$. KASUMI [5] is a 64-bit block cipher with a 128-bit key. The 64-bit inputs messages are generated by Eq. (6.3) with $n = 64$ and the component sequences and functions are obtained using Eq. (6.4) with $\mathrm{C}_j = \mathrm{KASUMI}\,(\mathrm{K}, \mathrm{R}_j)$.

6.3 Distinguishing Attack Model

In this section, we describe the attack model of our distinguishing attack on a message authentication code and a block cipher. In this paper we restrict ourselves to message authentication codes and block ciphers. The attack model is based on indistinguishability (IND) of encryptions under chosen-plaintext attack (CPA) (IND-CPA), which was first developed due to Goldwasser and Micali [55] in public-key settings. In [17], Bellare *et al.* studied the indistinguishability of encryptions under chosen-plaintext attack in the symmetric key setting. Here, we use the same attack model to distinguish MACs (or ciphertexts) in the symmetric-key setting. However, we develop a new distinguishing technique based on linear complexity of component sequences in the MOFM for deciding the MAC (or ciphertext). For the message authentication code, the aim of an adversary is to distinguish two MACs for two messages P_0 and P_1 with a high probability where messages P_0 and P_1 were chosen by the adversary. On the other hand, for an encryption, the adversary aims at distinguishing two ciphertexts for two chosen messages P_0 and P_1 with a high probability.

Let $F : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^m$ be a cryptographic algorithm which accepts two inputs, a key of length k and a message of length n and produces an output of length m. Assume that P_0 and P_1 are two messages of length n chosen by the adversary, the length of the key K is k and $c_i = F(K, P_i), i = 0, 1$. The aim of the distinguishing attack is to distinguish c_0 and c_1 for the messages P_0 and P_1 with high probability. We denote the oracle by $\mathcal{O}$ and the adversary by $\mathcal{A}$. The indistinguishability game [18, 55] between the oracle and the adversary is described as follows.

(1) Fixing a key K and generating the set of messages $\mathcal{R} = \{R_0, R_1, ..., R_{N-1}\}$ using an LFSR with a primitive polynomial of degree ℓ, $N = 2^\ell - 1$;

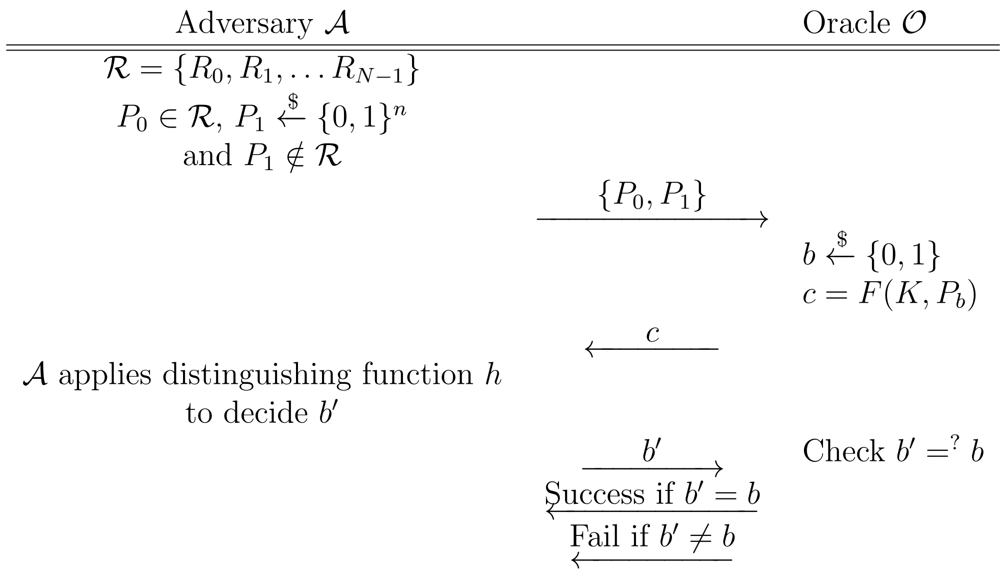
(2) The adversary $\mathcal{A}$ randomly picks up $P_0 \in \mathcal{R}$ and $P_1 \notin \mathcal{R}$ and sends both $\{P_0, P_1\}$ to $\mathcal{O}$.

(3) The oracle picks up $P_b \overset{\$}{\leftarrow} \{P_0, P_1\}$, $b = 0$ or 1 and computes $c = F(K, P_b)$. $\mathcal{O}$ sends c to the adversary $\mathcal{A}$.

(4) Once $\mathcal{A}$ receives c as a challenge, the adversary performs a technique and decides b' and returns b' to $\mathcal{O}$ where $b' = 0$ or 1;

(5) If $b = b'$, then adversary $\mathcal{A}$ **succeeds**; otherwise she **fails**.

We also summarize the game in Figure 6.1.



Adversary $\mathcal{A}$
Oracle $\mathcal{O}$
$\mathcal{R} = \{R_0, R_1, \dots R_{N-1}\}$
$P_0 \in \mathcal{R}$, $P_1 \overset{\$}{\leftarrow} \{0,1\}^n$ and $P_1 \notin \mathcal{R}$
$\{P_0, P_1\}$
$b \overset{\$}{\leftarrow} \{0,1\}$
$c = F(K, P_b)$
c
$\mathcal{A}$ applies distinguishing function h to decide b'
b'
Check $b' \overset{?}{=} b$
Success if $b' = b$
Fail if $b' \neq b$

Figure 6.1: Indistinguishability game

It is easy to see that, for a random cipher $\mathcal{B}$, the success rate of winning the game for an adversary is 1/2. In the following section, we present a new method to distinguish the MACs produced by f_1 for P_0 and P_1 with probability greater than 1/2. Therefore, the new method provides a construction of a distinguisher on f_1.

6.4 Distinguishing Attack Based on Linear Complexity

In this section, we first present a general technique to build a distinguisher of a cryptographic primitive, followed by the theoretical determination of the success probability of the distinguishing attack. In particular, we make use of the distribution of the linear complexity of component sequences of a primitive to develop a new distinguisher. Finally, we apply this technique on f_1, AES, KASUMI, and PRESENT.

6.4.1 A generic framework to build a distinguisher

We start this section by the following definition.

Definition 13. *Let $\mathcal{R}$ and $\mathcal{S}$ be two subsets of U, where $\mathcal{S} = U \setminus \mathcal{R}$. Let Ω be a subset of $\mathcal{R} \times \mathcal{S}$. Let $\mathcal{C}$ be a cryptographic scheme from U to some set V. For any $P_0 \in \mathcal{R}$ and $P_1 \in \mathcal{S}$, define a distinguishing function $h : \{\mathcal{C}(P_0), \mathcal{C}(P_1)\} \to \{0, 1\}$. We say that $\mathcal{C}$ is distinguishable with respect to $\mathcal{R}, \mathcal{S}, h, \Omega$ if the average probability*

$$\sum_{i \in \{0,1\}} \Pr\Big(h(c) = i \wedge c = \mathcal{C}(P_i)\Big)$$

is non-negligible compared with $1/2$, when (P_0, P_1) is randomly chosen from Ω.

Now we state the main theorem below and the proof of it.

Theorem 12. *Let the notations be the same as above. Now we define a subset $\mathcal{CS}$ of U, which is called the condition set. Let $\mathcal{S}' \subset \mathcal{S}$ and $\Omega = \mathcal{R} \times \mathcal{S}'$. For any $P_0 \in \mathcal{R}, P_1 \in \mathcal{S}'$, let us define the distinguishing function $h : \{\mathcal{C}(P_0), \mathcal{C}(P_1)\} \to \{0, 1\}$ as*

$$h(y) = \begin{cases} 0 & \text{if } y = \mathcal{C}(x) \text{ and } x \in \mathcal{CS}, \\ 1 & \text{otherwise.} \end{cases} \tag{6.7}$$

Define the following two probabilities

$$\begin{aligned} q_0 &= \Pr\left(x_0 \in \mathcal{R} \wedge x_0 \in \mathcal{CS}\right), \\ q_1 &= \Pr\left(x_1 \in \mathcal{S}' \wedge x_1 \in \mathcal{CS}\right). \end{aligned} \tag{6.8}$$

where $(x_0, x_1) \xleftarrow{\$} \Omega$. Then the average probability is

$$\sum_{i \in \{0,1\}} \Pr\left(h(c) = i \wedge c = \mathcal{C}(P_i)\ \right) = \frac{1 + (q_0 - q_1)}{2}. \tag{6.9}$$

Proof. It is not difficult to see that there are four independent cases of the event $h(c) = i \wedge c = \mathcal{C}(P_i)$ when $i \in \{0, 1\}$, therefore we may compute its probability one by one and sum them together:

(1). $h(c) = 0 \wedge c = \mathcal{C}(P_0) \wedge P_0 \in \mathcal{CS}$. The probability of this case equals

$$\Pr\left(h(c) = 0 \mid c = \mathcal{C}(P_0) \wedge P_0 \in \mathcal{CS}\ \right) \Pr\left(c = \mathcal{C}(P_0) \mid P_0 \in P\ \right) \Pr\left(P_0 \in \mathcal{CS}\ \right) = \frac{1}{2}q_0;$$

(2). $h(c) = 0 \wedge c = \mathcal{C}(P_0) \wedge P_0 \notin \mathcal{CS}$. The probability of this case is clear 0.

(3). $h(c) = 1 \wedge c = \mathcal{C}(P_1) \wedge P_0 \in \mathcal{CS}$. The probability of this case equals

$$\Pr\left(h(c) = 1 \mid c = \mathcal{C}(P_1) \wedge P_0 \in \mathcal{CS}\ \right) \Pr\left(c = \mathcal{C}(P_1) \mid P_0 \in P\ \right) \Pr\left(P_0 \in \mathcal{CS}\ \right) = \frac{1}{2}q_0(1-q_1).$$

(4). $h(c) = 1 \wedge c = \mathcal{C}(P_1) \wedge P_0 \notin \mathcal{CS}$. The probability of this case equals

$$\Pr\left(h(c) = 1 \mid c = \mathcal{C}(P_1) \wedge P_0 \notin \mathcal{CS}\ \right) \Pr\left(c = \mathcal{C}(P_1) \mid P_0 \notin P\ \right) \Pr\left(P_0 \notin \mathcal{CS}\ \right) = \frac{1}{2}(1-q_0)(1-q_1).$$

Summing the above probability we have the desired result

$$\sum_{i \in \{0,1\}} \Pr\left(h(c) = i \wedge c = \mathcal{C}(P_i)\ \right) = \frac{1 + (q_0 - q_1)}{2}.$$

The proof is completed. □

Several remarks on Theorem 12 are as follows:

(i) An attacker will expect the probability value in (6.9) to be as large as possible so that she can distinguish the cryptographic scheme $\mathcal{C}$ with a high probability.

(ii) The difficulty of finding the distinguishing attack described in Theorem 12 is to find a proper condition set $\mathcal{CS}$ such that $q_0 - q_1$ is large.

(iii) The value of $q_0 - q_1$ could be negative. If the attacker uses $\overline{\mathcal{CS}}$ to replace $\mathcal{CS}$, $q_0 - q_1$ will be positive, and the probability will be greater than 0.5. Thus, the problem of finding a condition set such that $q_0 - q_1$ is large becomes the problem of finding the condition set such that $|q_0 - q_1|$ is large.

(iv) In the rest of this section, we will show how to construct such set $\mathcal{CS}$, which leads to distinguishing attack on KASUMI and PRESENT with non-negligible success rate.

6.4.2 Distribution of the linear complexity of component sequences

We use f_1, AES,KASUMI and PRESENT as multi-output filtering functions and study the distribution of the linear complexities of their component sequences. Meidl and Niederreiter studied the expectation of the linear complexity of random binary periodic sequences in [83]. Unfortunately, the average values of the linear complexities of the component sequences of AES, f_1, KASUMI, PRESENT are very close to the theoretical value determined in [83] according to our experiments. This motivates us to look at the whole distribution of the linear complexity of the component sequences instead of considering only the average value. We perform the following test for the linear complexity and have an interesting observation on the component sequences of KASUMI and PRESENT.

Test of the distribution of linear complexity.

Usually, for a primitive $\mathcal{C}$, it is difficult to determine the distribution of linear complexity of its component sequences. Of course, one can choose a subset of inputs to the primitive to estimate the linear complexity distribution. However, since the input space is very large, it is hard to measure the accuracy of the estimated distribution. To avoid such problem, we propose a new method to test the distribution. This goal is achieved by choosing two (large) subsets of inputs and by comparing the distributions of the linear complexity of their component sequences. In particular, we choose one subset $\mathcal{LI}$ of the inputs to be generated by an ℓ-stage LFSR and the other subset $\mathcal{RI} = (\mathcal{LI} \setminus \{P_0\}) \cup \{P_1\}$, where $P_0 \overset{\$}{\leftarrow} \mathcal{LI}$ and $P_1 \overset{\$}{\leftarrow} \overline{\mathcal{LI}}$. Note that the elements in $\mathcal{LI}$ are ordered according to Eq. (6.3). It is clear that if the $\mathcal{C}$ has very good random property, it should not be easy to distinguish two distributions for $\mathcal{LI}$ and $\mathcal{RI}$. Our method consists of the following three steps.

Now fixing a primitive $\mathcal{C}$ and an ℓ-stage LFSR:

Step 1 (Generating component sequences).

We randomly choose N_{key} keys.

1. For all keys, using $\mathcal{LI}$ as the set of inputs and $\mathcal{C}$ as a multi-output filter, we obtain $m \cdot N_{key}$ component sequences. This set of component sequences is denoted by Q_1.
2. Similarly, using $\mathcal{RI}$ as the inputs, we generate another set of $m \cdot N_{key}$ component sequences, which is denoted by Q_2.

Step 2 (Computing linear complexity).

We compute the linear complexities of the sequences in Q_1 and Q_2 and count the number of component sequences in Q_i with the linear complexity $2^\ell - 2$ and $2^\ell - 1$, denoted by $N^i_{2^\ell-1}$ and $N^i_{2^\ell-2}$, where $i = 1$ or 2.

Step 3 (Comparing the distributions).

Now we compare two distributions by computing the slopes sl_i of the line between two points $(2^\ell - 2, N^i_{2^\ell-2})$ and $(2^\ell - 1, N^i_{2^\ell-1})$, where

$$sl_i = \frac{N^i_{2^\ell-1} - N^i_{2^\ell-2}}{(2^\ell - 1) - (2^\ell - 2)} = N^i_{2^\ell-1} - N^i_{2^\ell-2}.$$

If the difference between sl_1 and sl_2 is non-negligible, we can make use of it to build a distinguisher of $\mathcal{C}$, which is described in the next section. The worst case computational complexity for exhausting all ℓ-stage LFSRs of the above three steps is

$$\frac{\phi(2^\ell - 1)}{\ell} \times N_{\text{key}} \times 2\ell \times (2^\ell - 1) \times m, \tag{6.10}$$

where ϕ is the Euler phi function. We perform the experiment using these parameters on f_1, AES, KASUMI and PRESENT in the next section.

Distribution of f_1, AES, KASUMI and PRESENT.

In our experiment, we choose $\ell = 8$ and $N_{key} = 10^8$. By Eq. (6.10), the worst case complexity for the primitive f_1 is $2^{50.27}$ (some computation can be performed in a parallel way). We present the result in the following Figures In the figures, the red (resp. blue) line represents the distribution of sequences in Q_1 (resp. Q_2).

From Figures 6.2(a) and 6.2(b), one can observe that, for KASUMI and PRESENT, the difference of the distribution of the linear complexity for sequences in Q_1 and Q_2 is non-negligible. While Figures 6.2(c) and 6.2(d) show this is not the case for AES and f_1.

6.4.3 The new distinguishing attack

We now present the details of our distinguishing attack, which is achieved through constructing a distinguishing function h. The construction of the distinguishing function is

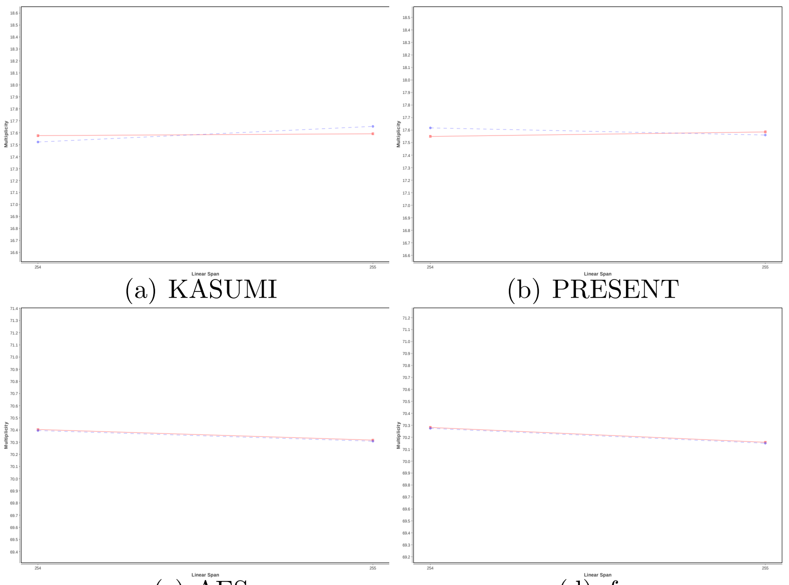

(a) KASUMI (b) PRESENT (c) AES (d) f_1

Figure 6.2: Distribution of the Component Sequences with Linear Complexity 254 and 255

based on the linear complexity distribution of the component sequences of a primitive in the MOFM.

Constructing the distinguishing function.

Recall that the distinguishing function is defined in Definition 13. We use the notations in Theorem 12 and the attack model is depicted in Figure 6.1.

1. Choosing an ℓ-stage LFSR with a primitive polynomial to generate the inputs of length n in $\mathcal{R}$ (see Eq. (6.3)). For f_1 and AES, $n = 128$; for KASUMI and PRESENT, $n = 64$.
2. Constructing $\mathcal{S} = \mathbb{F}_2^n \setminus \mathcal{R}$;
3. Randomly choose a message $P_0 \in \mathcal{R}$ and $P_1 \in \mathcal{S}$;
4. Let N_{LC} be the number of component sequences with linear complexity LC where $\ell \leq LC \leq 2^\ell - 1$;

5. Defining the condition set

$$\mathcal{CS} = \left\{ y \in \mathbb{F}_2^n \;\middle|\; \begin{array}{l} \text{using } (\mathcal{R} \setminus \{P_0\}) \cup \{y\} \text{ as the inputs of a primitive in the} \\ \text{MOFM, the slope of the line between} \\ \text{the points } (2^\ell - 2, N_{2^\ell - 2}) \text{ and } (2^\ell - 1, N_{2^\ell - 1}) \text{ is less than } t. \end{array} \right\};$$

6. The distinguishing function h is defined in Eq. (6.7) using the condition set $\mathcal{CS}$;

7. q_0, q_1 are the probability values defined in Definition 13.

6.4.4 An example of the attack

In this section, we apply the attack with our distinguishing function defined in Section 6.4.3 on f_1, AES, KASUMI, and PRESENT. For simplicity, we use an 8-stage LFSR to conduct our attack. However, one can use an arbitrary stage LFSR based on computation capability. For each of those four ciphers, we compute over 10^8 keys to obtain the distribution of the linear complexity. Figures 6.2(a) - 6.2(d) show the average number of the component sequences, which has the linear complexity 254 and 255. However, this computation was so heavy that it costed more than 4 weeks, even it was computed by an eighty-core server.

Theorem 12 and the observations in Figures 6.2(a) and 6.2(b) enable us to gain a non-negligible success rate of the attack on KASUMI and PRESENT. In the following, we present an example to show a possible application inspired by the observation shown in Figure 6.2 with 2^{10} keys.

We first choose an 8-stage LFSR to construct the set $\mathcal{R}$. We then randomly choose 2^{10} keys. For each key, a message $P_0 \in \mathcal{R}$ and message $P_1 \in \mathcal{S}$ are chosen randomly. In Figure 6.1, we use the distinguishing function h to execute the attack. It is worth to mention that, to test whether the average success rate is stable, we repeated the experiment 20 times by choosing different groups of 2^{10} keys and found similar results for most experiments. Due to the page limit, we present the largest success rate that we can achieve in Table 6.1, where we use the upper bound of the slope t and an 8-stage LFSR.

One can observe from the average success rate in Table 6.1 that the outputs of both KASUMI and PRESENT can be distinguished from a random primitive with a non-negligible probability. On the other hand, the performance of f_1 and AES is very similar to the random one.

Remark 4. *Note that in order to increase the accuracy of the success rates shown in Table 6.1, a larger scale experiment needs to be done, which will be designed as one of our future work.*

Table 6.1: Average success rate of our attack on f_1, AES, KASUMI and PRESENT

Primitive	t	q_0	q_1	Avg. Succ. Rate
f_1	2	0.20398	0.194458	50.476%
AES	2	0.193848	0.20044	50.329%
KASUMI	4	0.421875	0.454103	51.612%
PRESENT	5	0.5686	0.540285	51.416%

6.5 Distribution of the Algebraic Degree and Nonlinearity of the Component Functions

In this section, we investigate the distribution of the algebraic degree and the nonlinearity of the component functions of f_1, AES, KASUMI, and PRESENT in the MOFM. To measure the randomness property, we first determine the distribution of the algebraic degree and the nonlinearity of component functions using a random primitive as the multi-output filter. Comparing this ideal distribution with those of f_1, AES, KASUMI and PRESENT obtained by performing experiments, some non-randomness property of KASUMI is discovered. On the other hand, our experimental results show that f_1, AES and PRESENT perform very similar to the ideal case in the sense of the distributions of the algebraic degree and nonlinearity.

6.5.1 Algebraic degree distribution

Recall that the algebraic degree of a Boolean function is defined in Section 6.1. The following result states the number of Boolean functions with a given algebraic degree. The first part of the result can also be found in [33]. We provide a simple proof below for the completeness.

Theorem 13. *Let f be a Boolean function on $\mathbb{F}_{2^n}$. Then the number of Boolean functions with algebraic degree at most d is $2^{\sum_{i=0}^{d}\binom{n}{i}}$, and the number of Boolean functions with algebraic degree exactly d is $\left(2^{\binom{n}{d}}-1\right)2^{\sum_{i=0}^{d-1}\binom{n}{i}}$*

Proof. Denoting the set $\Omega = \{0, 1, \ldots, n-1\}$. Let the ANF of f be $f(x) = \sum_{I\in\mathcal{P}(\Omega)} a_I x^I$. If the degree of f is at most d, then all $a_I = 0$ for $|I| > d$. Clearly there are $\sum_{i=0}^{d}\binom{n}{i}$ terms in the ANF of f with $|I| \leq d$, and their coefficients can be either 0 or 1. Therefore there are $2^{\sum_{i=0}^{d}\binom{n}{i}}$ Boolean functions with degree at most d. For simplicity, let us denote by A_d the number of Boolean functions with degree at most d. Then by noting the number of Boolean functions with degree exactly d is $A_d - A_{d-1}$ we obtain the result.

□

Corollary 4. *Let $\mathcal{C}$ be a random cryptographic primitive and $\mathcal{L}$ be an n-stage LFSR whose characteristic polynomial is a primitive polynomial of degree n. We use $\mathcal{C}$ as a multi-output filtering function and $\mathcal{L}$ to generate the inputs of $\mathcal{C}$. Then the probability of the component functions having degree at most d is $\frac{2^{\sum_{i=0}^{d}\binom{n}{i}}}{2^{2n}}$. In particular, $Pr(d \leq n-3) = \frac{1}{2^{n+1}}$.*

Several remarks on the application of Theorem 13 are in the sequel:

(1) Assume the primitive $\mathcal{C}$ is used to generate MACs (for instance the function f_1 in TUAK). If the percentage of component functions with degree less than $n-2$ is large, then we may use the decoding method of the Reed-Muller code $R(n, n-3)$ to forge the MACs. See [80] for the Reed-Muller decoding. Note that the code $R(n, n-3)$ is the set of Boolean functions on $\mathbb{F}_{2^n}$ with algebraic degree at most $n-3$. Therefore, we need the probability $\Pr(d \leq n-3)$ to be as small as possible.

(2) On the other way, as shown in Corollary 4, for a random primitive, the probability $\Pr(d \leq n-3) = \frac{1}{2^{n+1}}$. So, for the primitive $\mathcal{C}$, if this probability is very different with $\frac{1}{2^{n+1}}$, some non-randomness properties may be exploited.

(3) The probability $\Pr(d \leq n-3)$ is actually affected by the diffusion property of the primitive $\mathcal{C}$. Assumed $\mathcal{C}$ is a keyed primitive from $\mathbb{F}_{2^n}$ to $\mathbb{F}_{2^m}$. In the modern design of ciphers, by increasing the number of iteration rounds, normally $\mathcal{C}$ could attain the maximal possible degree for any key K. For a keyed primitive $\mathcal{C}_K$, in the multi-output model, we restrict the inputs of $\mathcal{C}_K$ to a subspace $\mathcal{S}$ generated by an LFSR. For a fixed key K, $\mathcal{C}_K$ can be regarded as a vectorial function and the ANF of $\mathcal{C}_K$ has the form $\mathcal{C}_K(x) = \sum_{I\in\mathcal{P}(\Omega)} a_I(K)x^I$, where $\Omega = \{0, \ldots, n-1\}$ and $\mathcal{P}(\Omega)$ is the power set and $a_I(K) \in \mathbb{F}_{2^m}$ are the coefficients of x^I (a_I is a function with K as the variable) [33]. Then the restrictions of $\mathcal{C}_K|_{\mathcal{S}} = \sum_{I\in\mathcal{P}(\Omega), I\subset\mathcal{S}} a_I(K)x^I$. The degree d of the component functions is then determined by a_I with $|I| = d$. If the diffusion property of $\mathcal{C}$ and the key generating algorithm are good, it should be very rare that all $a_I = 0$ for $|I| \geq \dim(\mathcal{S}) - 2$.

To better understand Theorem 13 and the above comments, for f_1,AES, KASUMI and PRESENT, we perform the following test on the distribution of the algebraic degree of their component functions.

Statistical Test 1. *By Corollary 4, using an LFSR with a primitive polynomial of degree 8, the probability that the degree of the component functions is smaller than 7 is* $\frac{1}{2^9} = 19.53125 \times 10^{-4}$. *For* f_1, *AES, KASUMI and PRESENT, we apply the MOFM as in Section 6.2.1. We choose* 50,000 *keys for these primitives and compute the degree of the component functions. The probability of the degree is smaller than 7 is listed in the following table.*

Table 6.2: Distribution of the degree smaller than 7

Cryptographic primitive	$\Pr(d \le 6)$
Random function	19.53125×10^{-4}
f_1	19.87×10^{-4}
AES	19.77×10^{-4}
KASUMI	20.16×10^{-4}
PRESENT	19.58×10^{-4}

From Table 6.2, we can see that for KASUMI, the probability $Pr(d \le 6)$ *is much higher than the one for other ciphers. To confirm this, we test another 50000 keys and found the probability is very close to it. This points out a distinguisher of KASUMI and other ciphers in Table 6.2.*

6.5.2 Nonlinearity distribution

The nonlinearity of a Boolean function is one of the most important cryptographic properties. A highly nonlinear function is used to avoid the linear attack and its variants. Let f be a Boolean function on $\mathbb{F}_2^n$. The *nonlinearity* of f is defined in Section 6.1. One can see easily from its definition that, in other words,

$$\mathrm{NL}(f) = \max_{g \in \mathrm{RM}(1,n)} d(f, g),$$

where $\mathrm{RM}(1, n)$ denotes all Boolean functions with degree at most 1, and $d(f, g)$ is the weight of the sequence $(f(x) + g(x) : x \in \mathbb{F}_2^n)$. It is well known that when n is even the best nonlinearity a Boolean function may achieve is $2^{n-1} - 2^{n/2-1}$ and such functions are called bent functions (see [33] for more details). However, such functions are very rare. For a random Boolean function, we have the following result on the distribution of its nonlinearity.

Theorem 14 ([33, 100]). *Let c be any strictly positive real number. The density of the set*

$$\left\{ f \in \mathcal{B}_n,\ NL(f) \geq 2^{n-1} - c\sqrt{n}2^{\frac{n-1}{2}} \right\}$$

is greater than $1 - 2^{n+1-c^2 n \log_2 e}$. If $c^2 \log_2 e > 1$, then this density tends to 1 when n tends to infinity.

Applying the above theorem on Boolean functions with 8 variables, we have the following table. Note that the best nonlinearity we expect for Boolean functions with 8 variables is $2^7 - 2^3 = 120$.

Table 6.3: Lower bound of the density of Boolean functions in $\mathcal{B}_8$ with nonlinearity greater than W

Lower Bound W of NL	Lower bound of the density of Boolean functions with $\mathrm{NL}(f) \geq W$
98	0.5475
97	0.7190
96	0.8282
95	0.8966
94	0.9387
93	0.9642
92	0.9795
91	0.9884
90	0.9935

From the above table, one can see that if the component functions of f_1 are random, the probability that the component Boolean functions have nonlinearity smaller than 90 is very small, which is $1 - 0.99354511316750952827725848552 4 \approx 0.00645$. In view of this, we perform the following statistical test for f_1, AES, KASUMI and PRESENT.

Statistical Test 2. *Let the LFSR and the other settings be the same as in* **Statistical Test 1**. *We list the distribution of the nonlinearity of the component functions of f_1 and AES in the following table. Since only the component functions with smallest nonlinearity are important to us (as an attacker), we only list the probability that a Boolean function has nonlinearity smaller than 90 or 91. The notation $\Pr_{<W}$ denotes the probability that the nonlinearity is smaller than W.*

Table 6.4: The distribution of the nonlinearity of component sequences of f_1, AES, KASUMI and PRESENT

Cryptographic primitive	$\mathrm{Pr}_{<90}$	$\mathrm{Pr}_{<91}$
Random Function	0.006455	0.011597
f_1	0.000299	0.000690
AES	0.000306	0.000592
KASUMI	0.000299	0.000565
PRESENT	0.000308	0.000589

Unlike the distribution of the algebraic degree, from the above table we can not see obvious difference among these four ciphers. However, one can still see that the probability values $Pr_{<90}$ and $Pr_{<91}$ is still very different with the random case (although they are only the ***upper bounds*** *of the probability).*

Although now we cannot derive attacks from Statistical Test 1 and Statistical Test 2, it is interesting to observe some non-randomness in the aspect of the distribution of cryptographic properties.

6.6 Summary

In this Chapter, we introduced the MOFM for analyzing the security of a cryptographic primitive. In this model, a cryptographic primitive is used as a multi-output filtering function and a number of component sequences and component functions of the primitive are obtained. We aimed at exploiting the security properties of the primitive through studying its component sequences and functions.

Thanks to the fruitful research outcome in the theory of sequences and Boolean functions, we propose a general distinguish attack technique under IND-CPA. We developed a new object, called a distinguishing function, to characterize the success rate of our new attack method. Interestingly enough, for a primitive $\mathcal{C}$, by comparing the distribution of the linear complexity of the component sequences generated by two sets of inputs, we can construct a new distinguishing function. The importance of this new distinguishing function is demonstrated by launching an attack on KASUMI and PRESENT with non-negligible success rates.

Furthermore, we studied the cryptographic properties of the component functions. By comparing the distribution of the algebraic degree and nonlinearity properties with that of a random one, we discovered that, for KASUMI, its distribution of the algebraic degree is very different, while the distribution of f_1, AES and PRESENT is not. We cannot propose any immediate attack based on this observation, but it is interesting to point it out for future research.

Regarding to the future work, we believe it is important to study which inner structure of a primitive affects the distribution of the linear complexity, algebraic degree, nonlinearity, and other properties of component sequences and functions. This study may lead to a new attacking method, and present new criteria on designing a cryptographic primitive.

Chapter 7

Stream Cipher Based MACs Construction

In this chapter, we focus on constructing the MACs based on stream ciphers. As introduced in the preliminary chapter, EIA1 and EIA3 are two outstanding stream-cipher-based MACs, and they both have solid security proof and high efficiency. However, the cycling attack and the linear forgery attack make EIA1 and EIA3 not secure anymore. The cycling attack exploits the fact that the order of an element randomly selected from a finite field may be small. Attackers can conduct substitution forgery by simply switching two message blocks. The cycling attack can be applied to any polynomial-evaluation-based MACs only when the random numbers used in those MACs have small order, and attackers must guess the order. Thus, this attack is a probabilistically attack. Compared with the cycling attack, the linear forgery is more serious, because it is a deterministic attack. Some researchers may argue that since the linear forgery attack breaks the security assumption (one-time pad) of EIA1 and EIA3, this attack does not compromise the security of these two. But more researchers have realized that the one-time pad assumption of EIA1 and EIA3 is too strong. They even proposed a concept called "misusing-resistance". A misusing-resistant MAC is resistant against the linear forgery attack.

This chapter introduces two new kinds of MAC, which are both resistant to the linear forgery attack. The first MAC called WGIA-128 is a variant of EIA1. The addition over finite field used in EIA1 is replaced by the addition over ring. The second MAC called AMAC has two different constructions, which are both based on APN functions. AMAC also resists the cycling attack.

The rest of this chapter is organized as follows. Section 7.1 proposes a variant of EIA1, which is resistant to the linear forgery attack. Section 7.2 presents the APN-function-based MAC, AMAC, which resists both the linear forgery attack and the cycling attack. The last section concludes this chapter.

7.1 WGIA-128 and Security Analysis

This section introduces WGIA-128 algorithm and presents the security analysis of WGIA-128 under two attack models. WGIA-128, a variant of EIA1, is a MAC based on WG-16 stream cipher. Because only WG-16 can guarantee the randomness properties required by the security proof of WGIA-128, this MAC is specially designed as the integrity protection algorithm of WG-16.

7.1.1 Integrity Algorithm WGIA-128

The integrity algorithm WGIA-128 is a MAC function that maps an input message and an integrity key IK to a fixed-length MAC. The allowed bit length of a message is no greater than 2^{64}. The inputs and output of the algorithm are given in Tables 7.1 and 7.2, respectively.

Table 7.1: The Inputs of WGIA-128

Parameter	Size(bits)	Remark
COUNT	32	The counter $(\mathsf{COUNT}_{31}, \ldots, \mathsf{COUNT}_0)$
FRESH	32	The random number $(\mathsf{FRESH}_{31}, \ldots, \mathsf{FRESH}_0)$
BEARER	5	The bearer identity $(\mathsf{BEARER}_4, \ldots, \mathsf{BEARER}_0)$
DIRECTION	1	The direction of transmission $\mathsf{DIRECTION}_0$
IK	128	The integrity key $(\mathsf{IK}_{127}, \ldots, \mathsf{IK}_0)$
LENGTH	64	The length of the input message
M	LENGTH	The input bit stream $(\mathsf{M}_{\mathsf{LENGTH}-1}, \ldots, \mathsf{M}_0)$

Table 7.2: The Output of WGIA-128

Parameter	Size(bits)	Remark
MAC	32	The output bit stream $(\mathsf{MAC}_{31}, \ldots, \mathsf{MAC}_0)$

Initialization.

WG-16's parameters described in Chapter 2 are initialized using the inputs listed in Table 7.1 as follows:

$$\begin{aligned} K =&(\mathsf{IK}_{127}, \ldots, \mathsf{IK}_1, \mathsf{IK}_0), \\ IV =&(\mathsf{COUNT}_{31}, \ldots, \mathsf{COUNT}_1, \mathsf{COUNT}_0, \\ &\mathsf{FRESH}_{31}, \ldots, \mathsf{FRESH}_1, \mathsf{FRESH}_0, \\ &\mathsf{COUNT}_{31} \ldots, \mathsf{COUNT}_1, \mathsf{COUNT}_0 \oplus \mathsf{DIRECTION}_0, \\ &\mathsf{FRESH}_{31}, \ldots, \mathsf{FRESH}_{16} \oplus \mathsf{DIRECTION}_0, \ldots, \mathsf{FRESH}_0) \end{aligned}$$

After both K and IV are loaded, WG-16 runs under the initialization mode to scramble the input.

Key Stream Generation.

In key stream generation phase, WG-16 runs under the running mode to generate 96-bit key stream $z_0, \ldots, z_{95}$, where z_0 and z_{95} are the first and last bits of the key stream, respectively. We denote the 96-bit key stream generated in this step by $z = (z_{95}, \ldots, z_0)$. Let $H = (H_{31}, \ldots, H_0)$ be a random number that is selected uniformly from $\mathbb{F}_{2^{32}}$ and is independent of z.

MAC Generation

The 96-bit key stream generated in the last step is partitioned into three blocks P, Q and OTP, where $P = (z_{95}, \ldots, z_{64})$, $Q = (z_{63}, \ldots, z_{32})$ and $OTP = (z_{31}, \ldots, z_0)$ are three 32-bit blocks. Let $L = \lceil \mathsf{LENGTH}/32 \rceil + 3$ and M_i be the i-th bit of the message. Denote $B_i = (M_{32 \cdot i+31}, \ldots, M_{32 \cdot i})$ for $0 \leq i \leq L-5$, $B_{L-4} = (0, \ldots, 0, M_{\mathsf{LENGTH}-1}, \ldots, M_{32 \cdot (L-4)})$, $B_{L-3} = (\mathsf{LENGTH}_{31}, \ldots, \mathsf{LENGTH}_0)$, $B_{L-2} = (\mathsf{LENGTH}_{63}, \ldots, \mathsf{LENGTH}_{32})$ and $B_{L-1} = (H_{31}, \ldots, H_0)$. The MAC is then computed as follows:

1. Set the 32-bit value $T_0 = 0$;
2. Compute $T_{i+1} = (T_i \boxplus B_i) \boxtimes P$, for $0 \leq i \leq L-1$;
3. Compute $\mathsf{MAC} = (T_L \boxtimes Q) \oplus OTP$.

Note that in the above MAC generation process the multiplication (denoted by $\boxtimes$) is defined over the finite field $\mathbb{F}_{2^{32}}$ generated by the irreducible polynomial $r(x)$, whereas the addition (denoted by $\boxplus$) is computed over a ring $\mathbb{Z}_{2^{32}}$. In this paper, Steps 1 and 2 are called **GHASH+**, which enhances the security of the original polynomial hash (i.e., GHASH) [82]. The drawback of a polynomial hash comes from its linear structure. Hence, we have replaced the XOR operations in the original polynomial hash by the modular 2^{32} additions to form a semi-polynomial hash, which enables us to keep the the efficiency of a polynomial-like hash and avoid the linear structure simultaneously. Since the operations for generating a MAC are no longer linear, the linear forgery attack is not applicable.

7.1.2 Attack Models

The security of WGIA-128 is analyzed under two attack models.

Model $\mathcal{A}$

This attack model is adopted from McGrew and Viega's work [82], in which adversaries have the capability to access a MAC-generation oracle and a MAC-verification oracle only once. This one-time-access restriction describes the case that no misuse exists in the system, and almost guarantees the one-time pad used in Krawczyk's proof [73]. Note that the one-time pad is not fully guaranteed, because usually it is implemented by stream ciphers, which are analog to the one-time pad.

Model $\mathcal{B}$

This attack model simulates a scenario that misuse occurs. Compared with Model $\mathcal{A}$, adversaries can query the MAC generation oracle with the same (IV, H) at most twice. In [111], Wu and Gong showed that EIA1 is vulnerable to the linear forgery attack under Model $\mathcal{B}$. Since WGIA-128 replaces the XOR operations in EIA1 by the modular 2^{32} additions, it resists the linear forgery attack.

7.1.3 Ideal 32-Tuple Distribution Property of WG-16

Property 3 (Ideal 32-Tuple Distribution Property)**.** *For $1 \leq t \leq 32$, each non-zero t-tuples over $\mathbb{F}_2$ occurs exactly $2^{16\times 32-t}$ times and the zero t-tuple, $2^{16\times 32-t}-1$ times. In particular, each non-zero 32-tuples over $\mathbb{F}_2$ occurs $2^{15\times 32}$ times and the zero t-tuple, $2^{15\times 32}-1$ times.*

This property has been proved theoretically in [59], which implies the following lemma.

Lemma 10.

$$Pr[W = r] \approx \frac{1}{2^{32}},$$

where W is a 32-tuple generated by WG-16 and r is an arbitrary element in $\mathbb{F}_{2^{32}}$.

Lemma 10 guarantees that all the random numbers using in WGIA-128 are uniformly distributed.

7.1.4 Security of WGIA-128 under Model $\mathcal{A}$

Since the last block of the inputs of WGIA-128 (i.e., $B_{L-1} = H$) is chosen uniformly from $\mathbb{F}_{2^{32}}$, the following theorem holds for GHASH+.

Theorem 15. *GHASH+ is almost universal, which means*

$$Pr[T_k = r] = \frac{1}{2^{32}} + \epsilon \approx \frac{1}{2^{32}},$$

where $\epsilon = 1/2^{96}$.

Proof. Assume the last step of GHASH+ is $T_k = (T_{k-1} \boxplus \mathrm{H}) \boxtimes P$. To obtain $Pr[T_k = r]$, where r is an arbitrary element in $\mathbb{F}_{2^{32}}$, we consider the following two cases.

Case 1: When $r \neq 0$, we have

$$\begin{aligned} & Pr[(T_{k-1} \boxplus H)P = r] \\ &\quad = \sum_{p\neq 0} (Pr[(T_{k-1} \boxplus H)P = r | P = p] Pr[P = p]), \\ &\quad = \sum_{p\neq 0} \sum_{a} (Pr[T_{k-1} = a, H = rp^{-1} - a | P = p] \\ &\qquad\quad Pr[P = p]), \end{aligned}$$

$$= \sum_{p\neq 0}\sum_{a}(Pr[H = rp^{-1} - a|T_{k-1} = a, P = p]$$
$$Pr[T_{k-1} = a|P = p]Pr[P = p]).$$

P is generated by WG-16, which means from Lemma 10, it is uniformly distributed. H is assumed to be uniformly distributed. Since both H and P are uniformly distributed, and they are independent of each other, we obtain

$$Pr[H = rp^{-1} - a|T_{k-1} = a, P = p] = \frac{1}{2^{32}},$$
$$Pr[P = p] = \frac{1}{2^{32}}.$$

Using these two equations, we have

$$Pr[(T_{k-1} \boxplus H)P = r] = \frac{1}{2^{32}}\frac{1}{2^{32}}\sum_{p\neq 0}\sum_{a} Pr[T_{k-1} = a|P = p].$$

Noting that $\sum_a Pr[T_{k-1} = a|P = p] = 1$, we finally get

$$Pr[(T_{k-1} \boxplus H)P = r] = \frac{2^{32}-1}{(2^{32})^2}.$$

Case 2: When $r = 0$, we have

$$\begin{aligned} & Pr[(T_{k-1} \boxplus H)P = r] \\ & \quad = \sum_{p\neq 0}(Pr(T_{k-1} \boxplus H = 0|P = p]Pr[P = p]) + \\ & \quad\quad 1 \cdot Pr[P = 0], \\ & \quad = \frac{2^{32}-1}{(2^{32})^2} + \frac{1}{2^{32}}. \end{aligned}$$

We then obtain

$$\begin{aligned} Pr[T_k = r] &= Pr[T_k = r|r \neq 0]Pr[r \neq 0] + \\ & \quad Pr[T_k = r|r = 0]Pr[r = 0] \\ &= \frac{2^{32}-1}{(2^{32})^2}\frac{2^{32}-1}{2^{32}} + \frac{1}{2^{32}}\frac{1}{2^{31}} \approx \frac{1}{2^{32}}. \end{aligned}$$

Therefore, GHASH+ is almost universal. □

Let $D^j = (\mathbf{M}^j, point_0^j, \cdots, point_{2^{32}-1}^j, r^j)$, where $point_i^j = (P_i^j, \mathrm{H}_i^j)$, and $P_i^j, \mathrm{H}_i^j, \mathbf{M}^j, r^j \in \mathbb{F}_{2^{32}}$. $\mathcal{D}$ is the set that contains all the D^j satisfying the following four conditions:

(a) For all $i_1 \neq i_2$, $P_{i_1}^j \neq P_{i_2}^j$;

(b) For all $D^{j_1} \neq D^{j_2}$, there always exists at least one $point_i^{j_1} \in D^{j_1}$, but $point_i^{j_1} \notin D^{j_2}$;

(c) For all $D^{j_1} \neq D^{j_2}$, $r^{j_1} \neq r^{j_2}$;

(d) $\mathrm{GHASH}^+(M_j, point_i^j) = r^j$.

Let $|\mathcal{D}| = N$ be the size of $\mathcal{D}$. Each D^j represents that the polynomial $\mathrm{GHASH}^+(\mathbf{M}_j, point) = r^j$ passes all $point_i^j$, for $0 \leq i \leq 2^{32} - 1$. If an adversary knows the whole set $\mathcal{D}$, given $r \in \mathbb{F}_{2^{32}}$ and 2^{32} points with different Ps, she can find a message $\mathbf{M}$, such that $\mathrm{GHASH}^+(\mathbf{M}, point) = r$ passes all given points. According to the notation in Theorem 15, we keep using T_k to represent the GHASH+ value of a message $\mathbf{M}$, whose length is k. Then "$T_k = t_k$ passes some points" is equivalent to say "$\mathrm{GHASH}^+(\mathbf{M}, point) = t_k$ passes some points", where t_k is an element in $\mathbb{F}_{2^{32}}$.

Notice that due to the computational limit, an adversary can never get the whole $\mathcal{D}$. Assuming that she can get a subset $\mathcal{S}$ of $\mathcal{D}$ with size n, it is quite difficult for adversaries to get the GHASH+ value of a new message based on a set of known GHASH+ values and messages. Therefore, $\mathcal{S}$ is selected randomly from $\mathcal{D}$. We assume that each element in $\mathcal{S}$ is chosen from $\mathcal{D}$ equally likely.

A *u-match* means that given any 2^{32} different points and a random element t_k, $T_k = t_k$ passes u given points. If $u = 1$, finding a u-match is a determinate problem. If $u > 1$, finding u-match becomes a hard problem. We can only randomly choose a $T_k = t_k$ to test whether it passes u given points or not. The probability that we can find a u-match is given by the following lemma.

Lemma 11. *When $u > 1$, the probability of finding a u-match of the given points by randomly searching is $1/2^{32u}$.*

Proof. Since the space of message is infinite. Then we can consider that the probability of $T_k = t_k$ passing any 2^{32} points is equal. Finding a u-match means finding a $T_k = t_k$ that passes u of the given points. Then the probability is

$$
\begin{aligned}
&Pr[\text{finding a u-match of the given points}] \\
= &\binom{2^{32}}{u} \frac{(2^{64})^{2^{32}-u}}{(2^{64})^{2^{32}}} = \binom{2^{32}}{u} \frac{1}{2^{64u}} \approx \frac{1}{2^{32u}}.
\end{aligned}
$$

□

Lemma 12.
$$Pr[T_k = t_k | T'_k = t'_k] \approx \frac{1}{2^{32}}.$$

Proof. Given one P, the adversary can get a H under the condition $T_k = t_k$. Thus she gets 2^{32} different points. The probability of finding a 2^{32}-match in $\mathcal{S}$ is n/N. Otherwise, the probability of finding a u-match, where $1 < u < 2^{32}$, is given by Lemma 11. When $u = 1$, the probability is 1. Thus,

$$\begin{aligned} & Pr[T_k = t_k, T'_k = t'_k] \\ & \quad \approx \frac{n}{N} + \frac{N-n}{N}\left(\frac{1}{2^{32}} + \sum_{u=2}^{u=2^{32}-1} \frac{u}{2^{32}} \frac{1}{2^{32u}}\right), \\ & \quad = \frac{n}{N} + \frac{N-n}{N} \frac{1}{2^{32}}(1+\epsilon), \end{aligned}$$

where ϵ is a very small value compared with $\frac{1}{2^{32}}$. Because of the limitation of the computational resources, $n << N$. Therefore, $Pr[T_k = t_k | T'_k = t'_k] \approx 1/2^{32}$. □

Theorem 16. *GHASH+ is AXU, which means*
$$Pr[T_k \oplus T'_k = r] \approx \frac{1}{2^{32}}$$
where T_k and T'_k are two tags generated by GHASH+, and r is an arbitrary element in $\mathbb{F}_{2^{32}}$.

Proof. Assume that T_k and T'_k are two tags generated by GHASH+. Obviously, if T_k and T'_k are generated by two different pairs of IVs and Hs, we have $Pr[T_k \oplus T'_k = r] = 1/2^{32}$. However, in Model $\mathcal{A}$, the adversary is allowed to query the MAC verification oracle many times with the same IV. As a result, we must investigate $Pr[T_k \oplus T'_k = r]$ under the condition that T_k and T'_k are generated by the identical pair of IVs and Hs. We first have
$$Pr[T_k \oplus T'_k = r] = \sum_{t_k} Pr[T_k \oplus T'_k = r | T_k = t_k] Pr[T_k = t_k].$$
From the Lemma 12, we obtain
$$Pr[T_k \oplus T'_k = r] \approx \sum_{t_k} \frac{1}{2^{64}} = \frac{1}{2^{32}}.$$
□

Remark 5. *Since the GHASH+ is AXU, the original security proof of GCM [82] can be immediately applied to WGIA-128.*

7.1.5 Security of WGIA-128 under Model $\mathcal{B}$

When deploying the EIA1 and WGIA-128 in 4G-LTE networks, one should avoid the scenario described in Model $\mathcal{B}$. However, misusing the algorithms may lead to the reuse of IV and H in practice. We demonstrate that even in Model $\mathcal{B}$, WGIA-128 is still more secure than EIA1. Since the security of the components directly affects the security of the whole algorithm and the most important components of EIA1 and WGIA-128 are the hash functions, we first show that GHASH+ is more secure than GHASH under Model $\mathcal{B}$.

Remark 6. *Under Model $\mathcal{B}$, Lemma 12 shows even known one tag, the output of the generation oracle is still universal.*

Lemma 13.

$$Pr[T_k = t_k | T'_k = t'_k, T''_k = t''_k] \approx \frac{1}{2^{32}},$$

Proof. The proof is quite similar to the proof of Lemma 12. So we do not present the details here.

□

Theorem 17. *GHASH+ is AXU under Model $\mathcal{B}$, which means*

$$Pr[\lambda_1 T_k \oplus \lambda_2 T'_k \oplus T''_k = r] \approx \frac{1}{2^{32}}$$

where T_k and T'_k are two tags generated by GHASH+, and r is an arbitrary element in $\mathbb{F}_{2^{32}}$.

The proof of Theorem 17 is straightforward from Lemma 13. Thus we omit it here.

7.2 AMAC and Two Constructions

We call the MAC constructed upon APN functions AMAC. This section presents two different constructions of AMAC. Both constructions have the same assumption that the underlying cipher $\mathcal{C}$ maps the integrity key to key stream uniformly. Formally, if the key size is k and the length of the key stream generated by $\mathcal{C}$ using key K is len, $len \leq k$, there are 2^{k-len} different values of K, which can generate this key stream. When $len > k$, this key stream is generated by one K with probability 2^{k-len}.

7.2.1 Construction I

In Construction I, we use n, k and l to denote the length of the output, the key and the number of blocks in the message respectively. Note that the size of one block is the same as the length of the output.

We construct a MAC from $\mathbb{F}_2^* \times \mathbb{F}_2^k$ to $\mathbb{F}_2^n$ using an APN function $f(x) : \mathbb{F}_2^n \mapsto \mathbb{F}_2^n$ and an underlying cipher $\mathcal{C}$. We remark that the underlying cipher $\mathcal{C}$ is a stream cipher or a block cipher (keyed hash function) in counter mode. For a stream cipher or a block cipher in counter mode, there are two stages, the initializing phase and running phase, which are denoted as **Init** and **Gen** in the following pseudo code respectively. **Init(**K**,**N**,**$\mathcal{C}$**)** means initialize the cipher with integrity key and the nonce, and **Gen(**$\mathcal{C}$**, i)** means get the i-th n-bit block from the key stream generated by $\mathcal{C}$. Our construction takes four inputs. The first one is the message, followed by the length of this message in bit. The third input is the integrity key, and the last argument is the nonce.

Algorithm 3: MAC

```
Input: Message ($\overline{\mathbf{M}}$), Length of $\overline{\mathbf{M}}$ ($L$), Key ($K$), Nonce ($N$)
$len \leftarrow L/n$;
$res \leftarrow L \; mod \; n$;
$ret = 0$;
Init($K$,$N$,$\mathcal{C}$);
for $i \leftarrow 0 \ldots len - 1$ do
    $H_1 \leftarrow$ Gen($\mathcal{C}$, i);
    $ret \leftarrow ret + M_i \cdot H_1$;
end
$i \leftarrow len$;
if $res \neq 0$ then
    $H_1 \leftarrow$ Gen($\mathcal{C}$, i);
    $ret \leftarrow ret + (M_{len}||\overline{\mathbf{0}}) \cdot H_1$;
    $i \leftarrow i + 1$;
end
$H_1 \leftarrow$ Gen($\mathcal{C}$, i);
$ret \leftarrow ret + L \cdot H_1$;
$(OTP||H_0) \leftarrow$Gen($\mathcal{C}$, i);
$ret \leftarrow f(ret + H_0) + OTP$;
return $ret$;
```

Algorithm 3 demonstrates the computation of the tag. The message is partitioned into blocks, and each block M_i has n bits. If the length of $\overline{\mathbf{M}}$ is not a multiple of n, the last block is padded with zeros to make it a complete block, whose length is n. OTP, H_0 and $H_1 \in \mathbb{F}_2^n$ are three random numbers, which are generated by the underlying cipher $\mathcal{C}$. We write Algorithm 3 as a function for the purpose of analysis. First let us define an auxiliary function $g_K(x)$.

$$g_K(\overline{\mathbf{M}}) = \sum_{i=0}^{l} M_i \cdot sub - key_i, \tag{7.1}$$

where M_l is the length L represented as a field element. Then we define Algorithm 3 as

$$F_K(\overline{\mathbf{M}}) = f(g_K(\overline{\mathbf{M}}) + H_0) + OTP. \tag{7.2}$$

Theorem 18. *If $f(x) : \mathbb{F}_2^n \mapsto \mathbb{F}_2^n$ is an APN function, then the success probability of the substitution forgery attack against Construction I is upper bounded by*

$$P_S \leq \frac{1}{2^{n-1}} - \frac{1}{2^{2n}}.$$

Proof. Assume two messages

$$\begin{aligned} \overline{\mathbf{M}} &= [M_0, \cdots, M_{l-1}] \text{ and} \\ \overline{\mathbf{M}}' &= [M'_0, \cdots, M'_{l'-1}], \end{aligned}$$

where $M_i, M'_i \in \mathbb{F}_{2^n}$ are blocks of each message. $\overline{\mathbf{M}} \neq \overline{\mathbf{M}}'$. WLOG, assume the length of $\overline{\mathbf{M}}$ is no smaller than the length of $\overline{\mathbf{M}}'$, i.e. $l \geq l'$.

$$\begin{aligned} \mathcal{G} &= g_K(\overline{\mathbf{M}}) + g_K(\overline{\mathbf{M}}') \\ &= \sum_{i=0}^{l'} (M_i + M'_i) \cdot sub - key_i + \sum_{i=l'+1}^{l} M_i \cdot subkey_i. \end{aligned}$$

Let

$$g_i = \begin{cases} M_i + M'_i & , 0 \leq i \leq l' \\ M_i & , l' < i \leq l \end{cases}$$

Then we have

$$\mathcal{G} = \sum_{i=0}^{l} g_i \cdot K_i.$$

To prove $F_K(\overline{\mathbf{M}})$ is an AXU MAC, we need to compute the following probability.

$$\begin{aligned}&Pr_K[f(g_K(\overline{\mathbf{M}}) + H_0) + f(g_K(\overline{\mathbf{M}}') + H_0) = c] \\ &\quad = \sum_{b \in \mathbb{F}_{2^n}} Pr_K[f(g_K(\overline{\mathbf{M}}) + H_0) + f(g_K(\overline{\mathbf{M}}) + H_0 + b) = c, \mathcal{G} = b].\end{aligned}$$

Let us consider the equation

$$\mathcal{G} = \sum_{i=0}^{l} g_i \cdot K_i = b. \tag{7.3}$$

Since Eqn. (7.3) is a linear function of sub-keys, it is easy to show that, for any $b \in \mathbb{F}_{2^n}$, there are $l-1$ different sub-key sequences $\{K_0, \cdots, K_{l-1}\}$ such that Eqn. (7.3) holds.

Case 1 $b \neq 0$**:** Let $g_K(\overline{\mathbf{M}}) + H_0 = x$ and $\mathcal{G} = b$. By definition of APN function,

$$f(x) + f(x + b) = c \tag{7.4}$$

has at most two solutions of x for any $b \neq 0$. Since H_0 is independent from $g_K(\overline{\mathbf{M}})$ and $g_K(\overline{\mathbf{M}}')$, for any given $\overline{\mathbf{M}}$ and $\overline{\mathbf{M}}'$, there are at most two H_0 such that Eqn (7.4) holds. Thus, there are totally $2^{n(l-1)+1}$ different $(K_0, \cdots, K_{l-1}, H_0)$ tuples that satisfy Eqn. (7.4). Note that the length of the key stream generated by $\mathcal{C}$ is $n(l+1)$-bit.

When $n(l+1) \leq k$, each $(K_0, \cdots, K_{l-1}, H_0)$ tuple is mapped from $2^{k-n(l+1)}$ keys. Thus, $2^{n(l-1)+1}$ different $(K_0, \cdots, K_{l-1}, H_0)$ tuples are mapped from
$2^{n(l-1)+1} * 2^{k-n(l+1)}$ keys. Therefore,

$$Pr_K[f(g_K(\overline{\mathbf{M}}) + H_0) + f(g_K(\overline{\mathbf{M}}') + H_0) = c, \mathcal{G} = b] \leq \frac{1}{2^{2n-1}}.$$

When $n(l+1) > k$, each $(K_0, \cdots, K_{l-1}, H_0)$ tuple is mapped from a possible key with probability $s^{k-n(l+1)}$. As the same argument above, the probability

$$Pr_K[f(g_K(\overline{\mathbf{M}}) + H_0) + f(g_K(\overline{\mathbf{M}}') + H_0) = c, \mathcal{G} = b] \leq \frac{1}{2^{2n-1}}.$$

Case 2 $b = 0$**.** As the same argument in Case 1, there are $l-1$ different sub-key sequences $\{K_0, \cdots, K_{l-1}\}$ such that $\mathcal{G} = 0$. Whatever H_0 is,

$$f(g_K(\overline{\mathbf{M}}) + H_0) + f(g_K(\overline{\mathbf{M}}') + H_0)$$

is always zero. Thus, there are $2^{n(l-1)}$ different $(K_0, \cdots, K_{l-1}, H_0)$ tuples such that $\mathcal{G} = 0$. By the same argument in Case 1,

$$Pr_K[f(g_K(\overline{\mathbf{M}}) + H_0) + f(g_K(\overline{\mathbf{M}}') + H_0) = c, \mathcal{G} = 0] = \frac{1}{2^{2n}}.$$

Thus,

$$\begin{aligned} & Pr_K[f(g_K(\overline{\mathbf{M}}) + H_0) + f(g_K(\overline{\mathbf{M}}') + H_0) = c] \\ & = \sum_{b \in \mathbb{F}_{2^n}} Pr_K[f(g_K(\overline{\mathbf{M}}) + H_0) + f(g_K(\overline{\mathbf{M}}') + H_0) = c, \mathcal{G} = b] \\ & \leq \frac{1}{2^{n-1}} - \frac{1}{2^{2n}}. \end{aligned}$$

Therefore, $f(g_K(\overline{\mathbf{M}}) + H_0)$ is an ϵ-AXU hash family, and $F_K(\overline{\mathbf{M}})$ is an ϵ-AXU MAC, where $\epsilon \leq 2^{-n+1} - 2^{-2n}$. By Theorem 5 in [73], the probability of the substitution forgery attack on Construction I is no greater than $2^{-n+1} - 2^{-2n}$. □

7.2.2 Construction II

We construct a MAC from $\mathbb{F}_2^* \times \mathbb{F}_2^k$ to $\mathbb{F}_2^n$ using an APN function $f(x) : \mathbb{F}_{2^n} \mapsto \mathbb{F}_{2^n}$ and an underlying cipher $\mathcal{C}$. The four arguments are the same as defined in Construction I. As assumed before, $\mathcal{C}$ uniformly maps the key K to the key stream, two n-bit blocks generated by $\mathcal{C}$ are independent.

Algorithm 4 demonstrates the second construction. In the second last statement, the return value is masked by both H and OTP. It seems that H_0 is not necessary. But in the proof of the following theorem, we will show H_0 is indispensable. To prove the following theorem, we define Construction II as a function.

$$F_K(\overline{\mathbf{M}}) = g_K(\overline{\mathbf{M}}) + OTP,$$

where

$$g_K(\overline{\mathbf{M}}) = \sum_{i=0}^{l-1} f(M_i + K_i) + K_l.$$

Theorem 19. *If $f(x) : \mathbb{F}_2^n \mapsto \mathbb{F}_2^n$ is an APN function, then the success probability of the substitution forgery attack against Construction II is*

$$P_S \leq \frac{2}{2^n}.$$

Algorithm 4: MACII

```
Input: Message ($\overline{\mathbf{M}}$), Length of $\overline{\mathbf{M}}$ ($L$), Key ($K$), Nonce ($N$)
$len \leftarrow L/n$;
$res \leftarrow L\ mod\ n$;
$ret \leftarrow 0$;
Init($K$,$N$,$\mathcal{C}$);
for $i \leftarrow 0 \ldots len - 1$ do
    $H_1 \leftarrow$Gen($\mathcal{C}$, i);
    $ret \leftarrow ret + f(M_i + H_1)$;
end
$i \leftarrow len$;
if $res \neq 0$ then
    $H_1 \leftarrow$Gen($\mathcal{C}$, i);
    $ret \leftarrow ret + f((M_{len}||\overline{\mathbf{0}}) + H_1)$;
    $i \leftarrow i + 1$;
end
$H_1 \leftarrow$Gen($\mathcal{C}$, i);
$ret \leftarrow ret + f(L + H_1)$;
$(OTP||H_0) \leftarrow$Gen($\mathcal{C}$, i);
$ret \leftarrow ret + H_0 + OTP$;
return $ret$;
```

Proof. Assume

$$\begin{aligned}\overline{\mathbf{M}} &= \{M_0, \cdots, M_{l-1}\},\\ \overline{\mathbf{M}}' &= \{M'_0, \cdots, M'_{l'-1}\}\end{aligned}$$

are two messages.
Case 1 $l \neq l'$**:** WLOG, assume $l > l'$. We have

$$\begin{aligned} g_K(\overline{\mathbf{M}}) + g_K(\overline{\mathbf{M}}') =& \sum_{i=0}^{l'-1} \left(f(M_i + K_i) + f(M'_i + K_i)\right) + \\ & \sum_{i=l'}^{l-1} f(M_i + K_i) + K_{l'} + K_l. \end{aligned} \tag{7.5}$$

Let Q denote the summation of all the terms in Equ. (7.5) except K_l. Since K is uniformly mapped to tuple $(K_0, \cdots, K_l)$, and K_l is independent from Q. $\forall b \in \mathbb{F}_{2^n}$, we have

$$\begin{aligned} Pr_K[g_K(\overline{\mathbf{M}}) + g_K(\overline{\mathbf{M}}') = b] =& \sum_{a \in \mathbb{F}_{2^n}} Pr_K[K_l = b + a] Pr[Q = a] \\ =& Pr_K[K_l = b + a] = \frac{1}{2^n}. \end{aligned}$$

Case 2 $l = l'$**:** We have

$$g_K(\overline{\mathbf{M}}) + g_K(\overline{\mathbf{M}}') = \sum_{i=0}^{l-1} \left(f(M_i + K_i) + f(M'_i + K_i)\right).$$

Let

$$g_i = f(M_i + K_i) + f(M'_i + K_i), \text{ for } 0 \leq i < l.$$

Since all K_i $(0 \leq i < l)$ are independent, g_i $(0 \leq i < l)$ are independent as well. Therefore,

$$\begin{aligned} & Pr_K[g_K(\overline{\mathbf{M}}) + g_K(\overline{\mathbf{M}}') = b] \\ =& \sum_{(a_0, \cdots, a_{l-2}) \in \mathbb{F}_{2^n}^{l-1}} Pr_K[g_{l-1} = b + \sum_{i=0}^{l-2} a_i] Pr_K[g_0 = a_0, \cdots, g_{l-2} = a_{l-2}] \end{aligned}$$

By definition of APN functions, $\forall b \in \mathbb{F}_{2^n}$ and $(a_0, \cdots, a_{l-2}) \in \mathbb{F}_{2^n}^{l-1}$,

$$Pr_K[g_{l-1} = b + \sum_{i=0}^{l-2} a_i] \leq \frac{2}{2^n},$$

Therefore, $Pr_K[g_K(\overline{\mathbf{M}}) + g_K(\overline{\mathbf{M}}') = b] \leq 2^{1-n}$ Thus, $g_K(\overline{\mathbf{M}})$ is an ϵ-AXU hash function family, where $\epsilon \leq 2^{1-n}$. By Theorem 5 in [73], the probability of the substitution forgery attack on Construction II is no greater than 2^{1-n}. □

Let us consider removing H_0 in the second last statement. The upper bound of the probability in Case 1 would be $3/2^n$ if we removed H_0. An opponent can achieve this bound by selecting two messages as follows.

$$\begin{aligned} \overline{\mathbf{M}} &= \{M_0, \cdots, M_{l'-1}, M_{l'}, \cdots, M_{l-1}\} \text{ and} \\ \overline{\mathbf{M}}' &= \{M_0, \cdots, M_{l'-1}\}. \end{aligned}$$

The first l' terms of Eqn. (7.5) are cancelled. The rest terms are all three-to-one mappings. Each term has at most three different values of sub-keys to make this term equal a certain value. Thus, the total probability is $3/2^n$. Case 1 would be the worst case if we removed H_0. Then P_S is upper bounded by $3/2^n$.

7.2.3 Security Analysis

Remark 7. *Although the security level of each construction is $n-1$ (forgery probability is $O(2^{n-1})$), we still consider this level is enough, because this bound is for any length of messages. In other words, the bound will not change when the length of the messages increases. To decrease this bound, we can compute in a larger field and apply "secure truncation" [21] to the tag, the same as EIA1.*

Since two constructions are AXU MACs, they are resistant against most attacks. But for cycling attack [97] and linear forgery attack [111], the resistance is not straightforward. Therefore, we only consider these two attacks in the following.

Cycling Attack.

Cycling attack is a kind of attack that can be applied to all polynomial based MACs. The polynomial based MACs have a polynomial evaluation block, which is addressed as follows.

$$T = \sum_i M_i P^{i+1},$$

where M_i is a message block and P is a random number. It treats message as a polynomial over finite field and evaluates this polynomial at P. If the order of P is smaller than the length of the message in block, there exists at least one pair of $P^i = P^j, i \neq j$. Then the adversary can switch M_i and M_j without changing the MAC.

Both EIA1 and EIA3 are vulnerable to this attack. Since EIA1 is a kind of polynomial based MAC, it is straight forward EIA1 cannot resist such attack. EIA3 is equivalent to a polynomial based MAC. Thus, it is unsurprised that EIA3 also suffers this attack.

In our algorithm, the sub-key in each round plays a similar role of P^i in polynomial based MAC. However, since our sub-key is generated by a stream cipher each time, the same sub-key appears twice with negligible probability. Moreover, even the same sub-key appears, it is hard to tell the exact position of this sub-key. Thus, the adversary can hardly make an attack.

Linear Forgery Attack.

This attack was proposed by Wu and Gong on Wisec13'. Because of the linear structure, known 2 pairs of message and tag pairs generated by EIA1, the adversary can forge up to 2^{32} message and tag pairs. This attack can be applied to EIA3 as well.

The two constructions in this paper are resistant to this attack, because the sub-key is generated by a stream cipher. The structure is no longer linear.

7.3 Implementation and Efficiency

In this section, we present some consideration regarding the implementation. At the end of this section, we compare our algorithms with EIA1 and EIA3, two MACs deployed in the 4G LTE system.

7.3.1 Selection of Fields and APN Function

The length of a MAC tag is usually some power of two. Several decades ago, people believed that 32-bit tag was enough. For a 32-bit tag, the complexity of the birthday attack is 2^{16}, which is too huge to be computed at that time. Therefore, the tag sizes of some legacy systems are still 32-bit, such as cellular, Wi-Fi, and etc. Since the computers become more powerful, 2^{16} is easy to compute even for a personal laptop computer. Usually, a powerful server has the capability to do the exhaustive search beyond 2^{48}. Therefore, nowadays even a 64-bit tag is still vulnerable to the birthday attack. People now increase the tag size to

128 or 256, for example TUAK [53]. We want our algorithm to work with both the legacy systems and the modern systems. For this reason, our design has four versions, 32-bit, 64-bit, 128-bit and 256-bit.

Selection of Finite Fields.

The multiplication over finite filed is a necessary block of our algorithm. Assume the defining polynomial of the finite field $GF(2^n)$ over $GF(2)$ is

$$f(x) = x^n + g(x).$$

To make the multiplication more efficient, we want both the degree and the number of terms of $g(x)$ to be as small as possible. Usually, for a USIM card, there is an 8-bit chip inside, which means it can compute 8-bit XOR simultaneously. The 8-bit platform is currently the smallest platform considered by us. Therefore, we restrict the degree of $g(x)$ to be smaller than eight. By exhaustive search, we find the defining polynomial for each version of our design. The defining polynomials are listed in Table 7.3. Note that for the field $GF(2^{256})$, we cannot find a polynomial satisfies our criterion. Thus, we loosen our condition, and find a polynomial that the degree of $g(x)$ is ten. It is not efficient on the 8-bit platform, because the XOR is computed in two clock cycles. But it is still efficient on the 16-bit and higher platforms.

Table 7.3: Defining Polynomials

Version	Field	Defining Polynomial
32-bit	$GF(2^{32})$	$x^{32} + x^7 + x^6 + x^2 + 1$
64-bit	$GF(2^{64})$	$x^{64} + x^4 + x^3 + x + 1$
128-bit	$GF(2^{128})$	$x^{128} + x^7 + x^2 + x + 1$
256-bit	$GF(2^{256})$	$x^{256} + x^{10} + x^9 + x^8 + x^7 + x^4 + x^2 + x + 1$

Selection of APN Function.

Another critical block is the APN function. There are several constructions of APN functions. Among all those constructions, we want the function has the following properties.

- Work in the field $GF(2^n)$, where n is even;

- Be efficient to compute.

Proposition 6 in [35] suggests one construction that has the form of

$$x^3 + \alpha Tr(\beta x^3 + \gamma x^9),$$

where $\alpha, \beta, \gamma \in GF(2^n)$, $\alpha \neq 0$ and n is even. We know that this construction is EA-equivalent to x^3. Since the computation of x^3 is more efficient than $x^3 + \alpha Tr(\beta x^3 + \gamma x^9)$, we choose x^3 as our function.

If the field element is represented under a normal basis, the square is simply cyclic shifting one bit. To compute x^3, we may first compute x^2, and then compute $x^2 \cdot x$.

7.3.2 Experiment Result of Efficiency

Since our MACs are based on APN function, we call it AMAC. We implement both Construction I and Construction II, which are called AMAC I and AMAC II respectively.

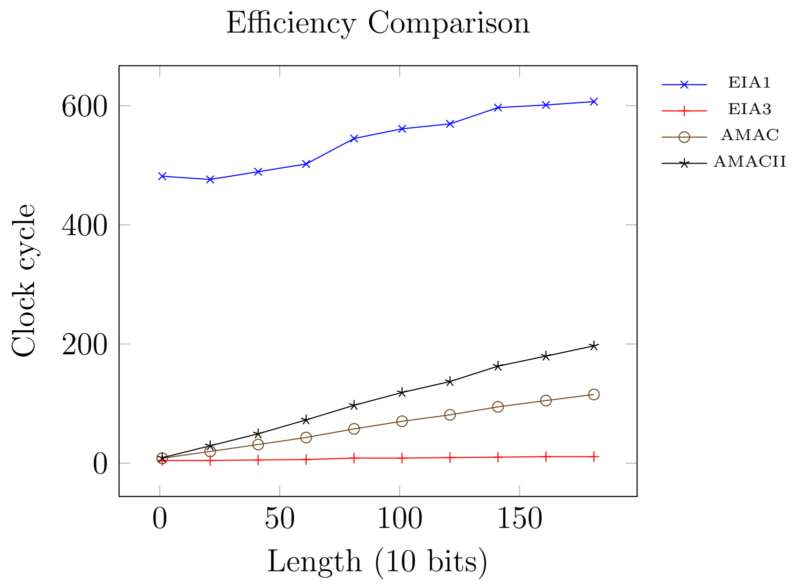


Figure 7.1: Efficiency comparison among EIA1, EIA3, and AMAC

Figure 7.1 shows the comparison of EIA1, EIA3, and our algorithms. In this test, we choose ZUC as our underlying cipher. From the figure we can clearly see that although our algorithms are slower than EIA3, they are overwhelmingly faster than EIA1.

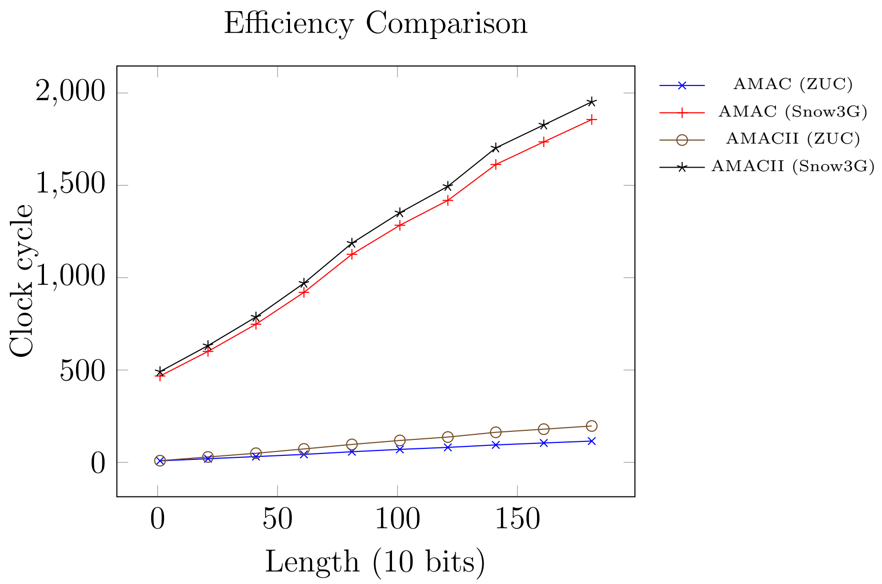


Figure 7.2: Efficiency comparison when using ZUC and Snow3G as underlying ciphers

Since we generate random numbers in each round, the efficiency of our algorithms highly depend on the underlying cipher. Figure 7.2 demonstrates the difference between different underlying ciphers. Obviously, ZUC is much faster than Snow3G.

7.4 Summary

We improved EIA1 such that the new variant is resistant to the linear forgery attack. After that, we proposed two new MAC constructions using APN function. Compared with previous works based on APN function, ours can take messages with any length as the input, and output a fixed-length tag as MAC. Such design is more flexible and practical. Both constructions are analog to XOR-MAC. However, both constructions have simpler round functions compared with XOR-MAC. Therefore, we have better performance than MACs based on block ciphers or keyed hash functions. Moreover, we have solid mathematical security proofs for both constructions. We compare the security and efficiency of our algorithm with two well known stream cipher based MACs, EIA1 and EIA3, which are deployed in 4G LTE cellular network. Our algorithms resist to the cycling attack and linear forgery attack, which can be applied to EIA1 and EIA3. The experiments show that our algorithms are slower than EIA3 but faster than EIA1.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

This thesis first proposes a practical attack on EIA1 called linear forgery attack, which exploits the linear structure of EIA1. Known two message-MAC pairs, the opponent can forge up to 2^{32} valid pairs.

Inspired by the linear forgery attack, this thesis assesses the security of EIA1 under different models. The security proof of EIA1 is only a special case of the analysis in this thesis. The assessment shows that when the assumption of one-time pad is broken, EIA1 is not safe at all. In addition to the linear forgery attack, attackers can even recover the key stream used in EIA1. Broking the one-time pad assumption is sometimes referred to as misuse. Several researchers have begun to study the misuse-resistant MACs, which have some level of security when misuse happens. After the security assessment, several optimizations of EIA1 are presented, and a more efficient polynomial evaluation algorithm is introduced to replace the Horner's Rule. The experiments suggest that the optimized version is much faster than the official implementation of EIA1.

For TUAK, this thesis suggests to use different constant INSTANCE for function f_2 - f_5 to follow the recommendation of Keccak. After this modification, we theoretically prove that f_1, f_1^* and f_2 are secure MACs, and f_3, f_4, f_5 and f_5^* are secure KDFs. Specifically, f_1, f_1^* and f_2 resist the universal key recovery, substitution forgery, pre-image and second pre-image attacks, and f_3, f_4, f_5 and f_5^* resist universal key recovery, pre-image and second pre-image attacks.

To assess TUAK, a new crypto analysis method called Multi-Output Filtering Model is proposed. This model is inspired by the system test using LFSR in communication systems. An LFSR generates the input, and the cipher works as the filtering function. The output can be treated as a sequence or the output of the boolean function. MOFM reveals the randomness of the cryptographic primitive being tested. This thesis applies MOFM to TUAK, AES, PRESENT and KASUMI. The results suggest that TUAK and AES have very good randomness properties, but PRESENT and KASUMI show some non randomness.

After studying existing MACs, such as EIA1, EIA3, TUAK, this thesis proposes two novel MAC constructions. The first called WGIA-128 is a variant of EIA1. The addition over finite filed used in EIA1 is replaced by the addition over ring in order to resist the linear forgery attack. The second called AMAC is quite different from EIA1 and EIA3, and it is based on APN functions. In addition to the linear forgery attack, AMAC is also resistant to the cycling attack.

8.2 Future Work

The work presented in this thesis is still improvable. We can think of several possible improvements of these works.

The linear forgery attack highly depends on the misuse of network operators. Our ultimate goal is to find a way to conduct the linear forgery attack without the misuse of network operators. In other words, we want to find a method, which follows the standard, to repeat IV and IK. To achieve the goal, we need carefully study the standard.

The theoretical explanation of MOFM is still unclear. Currently, we cannot explain the different performances between the strong group (AES and TUAK) and weak group (PRESENT and KASUMI). The difference is probably due to the algebraic degree or the nonlinearity of the filtering function. We conjecture that the MOFM test could be equivalent to some existing properties of boolean functions. If our conjecture is correct, the MOFM test could be an effective way to test those properties.

Although AMAC has better security than EIA1 and EIA3, the performance is not satisfactory. In the future, our goal is to design a faster MAC with security proof and misuse-resistant property. We believe the LFSR together with a simple nonlinear filtering function would be a good choice. For example, stream cipher in scrambling mode could be a MAC algorithm. But the security proof of scrambling mode is still an open question.

References

[1] Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2. Document 5: Design and Evaluation Report. Technical Report TR 35.919, 3^{rd} Generation Partnership Project.

[2] Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2. Document 2: SNOW 3G Specification. Technical Report TS 35.216, 3^{rd} Generation Partnership Project, September 2006.

[3] Specification of The 3GPP Confidentiality and Integrity Algorithms UEA2& UIA2. Document 1: UEA2 and UIA2 Specification. Technical Report TS 35.215, 3^{rd} Generation Partnership Project, 2006.

[4] Specification of the 3GPP Confidentiality and Integrity Algorithms. Document 1: f8 and f9 Specification. Technical Report TS 35.201, 3^{rd} Generation Partnership Project, 2007.

[5] Specification of the 3GPP Confidentiality and Integrity Algorithms. Document 2: Kasumi Specification. Technical Report TS 35.202, 3^{rd} Generation Partnership Project, June 2007.

[6] Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3. Document 1: 128-EEA3 and 128-EIA3 Specification. Technical Report TS 35.221, 3^{rd} Generation Partnership Project, January 2011.

[7] Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3. Document 2: ZUC Specification. Technical Report TS 35.222, 3^{rd} Generation Partnership Project, January 2011.

[8] 3G Security Security architecture. Technical Report TS 33.102, 3^{rd} Generation Partnership Project, December 2012.

[9] Mohamed Ahmed Abdelraheem, Peter Beelen, Andrey Bogdanov, and Elmar Tischhauser. Twisted Polynomials and Forgery Attacks on GCM. In *Advances in Cryptology–EUROCRYPT 2015*, pages 762–786. Springer, 2015.

[10] Martin Agren, Martin Hell, Thomas Johansson, and Willi Meier. Grain-128a: A New Version of Grain-128 with Optional Authentication. *Int. J. Wire. Mob. Comput.*, 5(1):58–59, December 2011.

[11] Mufeed ALMashrafi, Harry Bartlett, Leonie Simpson, Ed Dawson, and Kenneth Koon-Ho Wong. Analysis of Indirect Message Injection for MAC Generation Using Stream Ciphers. In *Information security and privacy*, pages 138–151. Springer, 2012.

[12] Jean-Philippe Aumasson and Willi Meier. Zero-sum Distinguishers for Reduced Keccak-f and for the Core Functions of Luffa and Hamsi. *rump session of Cryptographic Hardware and Embedded Systems-CHES*, 2009:67, 2009.

[13] Harry Bartlett, Mufeed AlMashrafi, Leonie Simpson, Ed Dawson, and Kenneth Koon-Ho Wong. A General Model for MAC Generation Using Direct Injection. In *Information Security and Cryptology*, pages 198–215. Springer, 2013.

[14] Mihir Bellare. New Proofs for NMAC and HMAC: Security without Collision-resistance. In *Proceedings of the 26th annual international conference on Advances in Cryptology*, CRYPTO'06, pages 602–19, Berlin, Heidelberg, 2006. Springer-Verlag.

[15] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying Hash Functions for Message Authentication. In *Advances in Cryptology—CRYPTO'96*, pages 1–15. Springer, 1996.

[16] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying Hash Functions for Message Authentication. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '96, pages 1–15, London, UK, UK, 1996. Springer-Verlag.

[17] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A Concrete Security Treatment of Symmetric Encryption. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 394–403. IEEE, 1997.

[18] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations Among Notions of Security for Public-Key Encryption Schemes. In *Advances in Cryptology—CRYPTO'98*, pages 26–45. Springer, 1998.

[19] Mihir Bellare, Roch Guérin, and Phillip Rogaway. XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions. In *Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '95, pages 15–28, London, UK, UK, 1995. Springer-Verlag.

[20] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The Security of The Cipher Block Chaining Message Authentication Code. *Journal of Computer and System Sciences*, 61(3):362–399, 2000.

[21] Daniel J Bernstein. Stronger Security Bounds for Wegman-Carter-Shoup Authenticators. In *Advances in Cryptology—EUROCRYPT 2005*, pages 164–180. Springer, 2005.

[22] Daniel J Bernstein. Second Ppreimages for 6 (7?(8??)) Rounds of Keccak. *NIST mailing list*, 2010.

[23] Guido Bertoni, Joan Daemen, Michaël Peeters, and GV Assche. The Keccak Reference. *Submission to NIST (Round 3)*, 13, 2011.

[24] Guido Bertoni, Joan Daemen, Michael Peeters, and GV Assche. Permutation-Based Encryption, Authentication and Authenticated Encryption. *Directions in Authenticated Ciphers (July 2012)*, 2012.

[25] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak Sponge Function Family Main Document. *Submission to NIST (Round 2)*, 3:30, 2009.

[26] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sakura: A Flexible Coding for Tree Hashing. In *Applied Cryptography and Network Security*, pages 217–234. Springer, 2014.

[27] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-Like Cryptosystems. *Journal of CRYPTOLOGY*, 4(1):3–72, 1991.

[28] John Black and Phillip Rogaway. CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '00, pages 197–215, London, UK, UK, 2000. Springer-Verlag.

[29] Andrey Bogdanov, Lars R Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew JB Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. *PRESENT: An Ultra-Lightweight Block Cipher*. Springer, 2007.

[30] Christina Boura and Anne Canteaut. Zero-Sum Distinguishers for Iterated Permutations and Application to Keccak-f and Hamsi-256. In *Selected Areas in Cryptography*, pages 1–17. Springer, 2011.

[31] Christina Boura, Anne Canteaut, and Christophe De Canniere. Higher-Order Differential Properties of Keccak and Luffa. In *Fast Software Encryption*, pages 252–269. Springer, 2011.

[32] Gilles Brassard. On Computationally Secure Authentication Tags Requiring Short Secret Shared Keys. In *Advances in Cryptology*, pages 79–86. Springer, 1983.

[33] Claude Carlet. Boolean Functions for Cryptography and Error Correcting Codes. *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, 134:257, 2010.

[34] Claude Carlet, Cunsheng Ding, and Harald Niederreiter. Authentication Schemes from Highly Nonlinear Functions. *Designs, Codes and Cryptography*, 40(1):71–79, 2006.

[35] Claude Carlet, Guang Gong, and Yin Tan. Quadratic Zero-Difference Balanced Functions, APN Functions and Strongly Regular Graphs. *Designs, Codes and Cryptography*, pages 1–26, 2014.

[36] J Lawrence Carter and Mark N Wegman. Universal Classes of Hash Functions. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 106–112. ACM, 1977.

[37] Samuel Chanson, Cunsheng Ding, and Arto Salomaa. Cartesian Authentication Codes from Functions with Optimal Nonlinearity. *Theoretical Computer Science*, 290(3):1737–1752, 2003.

[38] Lidong Chen and Guang Gong. *Communication System Security*, pages 358–359. CRC Press Taylor & Francis Group, 2012.

[39] Scott Contini and Yiqun Lisa Yin. Forgery and Partial Key-Recovery Attacks on HMAC and NMAC Using Hash Collisions. In *Advances in Cryptology–ASIACRYPT 2006*, pages 37–53. Springer, 2006.

[40] Joan Daemen and Vincent Rijmen. The Design of Rijndael: AES. *The Advanced Encryption Standard*, 2002.

[41] Joan Daemen and Gilles Van Assche. Differential Propagation Analysis of Keccak. In *Fast Software Encryption*, pages 422–441. Springer, 2012.

[42] Ivan Bjerre Damgård. A Design Principle for Hash Functions. In *Advances in Cryptology—CRYPTO'89 Proceedings*, pages 416–427. Springer, 1990.

[43] Cunsheng Ding and Harald Niederreiter. Systematic Authentication Codes from Highly Nonlinear Functions. *IEEE Transactions on Information Theory*, 50(10):2421–2428, 2004.

[44] Itai Dinur, Orr Dunkelman, and Adi Shamir. New Attacks on Keccak-224 and Keccak-256. In *Fast Software Encryption*, pages 442–461. Springer, 2012.

[45] Itai Dinur, Orr Dunkelman, and Adi Shamir. Collision Attacks on Up to 5 Rounds of SHA-3 Using Generalized Internal Differentials. In *Fast Software Encryption*, pages 219–240. Springer, 2014.

[46] Itai Dinur, Pawel Morawiecki, Josef Pieprzyk, Marian Srebrny, and Michal Straus. Practical Complexity Cube Attacks on Round-Reduced Keccak Sponge Function. Technical Report IACR 2014/259, International Association for Cryptologic Research, 2014.

[47] Itai Dinur and Adi Shamir. Cube Attacks on Tweakable Black Box Polynomials. In *Advances in Cryptology–EUROCRYPT 2009*, pages 278–299. Springer, 2009.

[48] Ming Duan and XueJia Lai. Improved Zero-Sum Distinguisher for Full Round Keccak-f Permutation. *Chinese Science Bulletin*, 57(6):694–697, 2012.

[49] Alexandre Duc, Jian Guo, Thomas Peyrin, and Lei Wei. Unaligned Rebound Attack: Application to Keccak. In *Fast Software Encryption*, pages 402–421. Springer, 2012.

[50] Orr Dunkelman, Nathan Keller, and Adi Shamir. A Practical-time Related-key Attack on the KASUMI Cryptosystem Used in GSM and 3G Telephony. In *Proceedings of the 30th Annual Conference on Advances in Cryptology*, CRYPTO'10, pages 393–410, Berlin, Heidelberg, 2010. Springer-Verlag.

[51] Michele Elia, Joachim Rosenthal, and Davide Schipani. Polynomial Evaluation over Finite Fields: New Algorithms and Complexity Bounds. *Applicable Algebra in Engineering, Communication and Computing*, 23(3-4):129–141, 2012.

[52] Petrank Erez and Rackoff Charles. CBC MAC for Real-Time Data Sources. *J. Cryptology*, 13:315–338, 2000.

[53] ETSI/SAGE. Specification of the TUAK Algorithm Set: A Second Example Algorithm Set for the 3GPP Authentication and Key Generation Functions $f_1, f_1^*, f_2, f_3, f_4, f_5$ and f_5^*, SP-130602. Technical Report TS 35.231, 3^{rd} Generation Partnership Project, September 2014.

[54] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. *Cryptography Engineering: Design Principles and Practical Applications.* John Wiley & Sons, 2011.

[55] Shafi Goldwasser and Silvio Micali. Probabilistic Encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984.

[56] Solomon W. Golomb. *Shift Register Sequences*, volume 78. Aegean Park Press Laguna Hills, CA, 1982.

[57] Solomon W. Golomb and Guang Gong. *Signal Design for Good Correlation For Wireless Communication, Crypography, and Radar.* Cambridge University Press, 2005.

[58] Guang Gong, Kalikinkar Mandal, Yin Tan, and Teng Wu. Security Assessment of TUAK Algorithm Set. 2014.

[59] Guang Gong and Amr M Youssef. Cryptographic properties of the Welch-Gong transformation sequence Generatorsnerators. *Information Theory, IEEE Transactions on*, 48(11):2837–2846, 2002.

[60] B Guido, D Joan, P Michaël, and VA Gilles. Cryptographic Sponge Functions. 2011.

[61] Raja Zeshan Haider. Birthday Forgery Attack on 128-EIA3 Version 1.5. Technical Report IACR 2011/268, International Association for Cryptologic Research, 2011.

[62] Helena Handschuh and Bart Preneel. Key-recovery attacks on universal hash function based mac algorithms. In *Advances in Cryptology–CRYPTO 2008*, pages 144–161. Springer, 2008.

[63] Shoichi Hirose and Hidenori Kuwakado. A scheme to base a hash function on a block cipher. In *Selected Areas in Cryptography*, pages 262–275. Springer, 2009.

[64] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. AEZ v1: Authenticated-Encryption by Enciphering. *CAESAR 1st Round, competitions. cr. yp. to/round1/aezv1. pdfl*, 2014.

[65] Robert V Hogg, Joseph McKean, and Allen T Craig. *Introduction to Mathematical Statistics*. Pearson Education, 2005.

[66] Ekawat Homsirikamol, Paweł Morawiecki, Marcin Rogawski, and Marian Srebrny. Security Margin Evaluation of SHA-3 Contest Finalists Through SAT-Based Attacks. In *Computer Information Systems and Industrial Management*, pages 56–67. Springer, 2012.

[67] William G Horner. A New Method of Solving Numerical Equations of All Orders, by Continuous Approximation. *Philosophical Transactions of the Royal Society of London*, pages 308–335, 1819.

[68] Tetsu Iwata, Keisuke Ohashi, and Kazuhiko Minematsu. Breaking and Repairing GCM Security Proofs. In *Advances in Cryptology–CRYPTO 2012*, pages 31–49. Springer, 2012.

[69] Gregory A Kabatianskii, Ben Smeets, and Thomas Johansson. On the Cardinality of Systematic Authentication Codes via Error-Correcting Codes. *Information Theory, IEEE Transactions on*, 42(2):566–578, 1996.

[70] Emilia Käsper and Peter Schwabe. Faster and Timing-Attack Resistant AES-GCM. In *Cryptographic Hardware and Embedded Systems-CHES 2009*, pages 1–17. Springer, 2009.

[71] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. CRC Press, 2014.

[72] Edwin Key. An Analysis of the Structure and Complexity of Nonlinear Binary Sequence Generators. *Information Theory, IEEE Transactions on*, 22(6):732–736, 1976.

[73] Hugo Krawczyk. LFSR-based Hashing and Authentication. In *Advances in Cryptology—CRYPTO'94*, pages 129–139. Springer, 1994.

[74] Hugo Krawczyk, Ran Canetti, and Mihir Bellare. HMAC: Keyed-hashing for Message Authentication. 1997.

[75] Kaoru Kurosawa and Tetsu Iwata. TMAC: Two-Key CBC MAC. In *Proceedings of the 2003 RSA conference on The cryptographers' track*, CT-RSA'03, pages 33–49, Berlin, Heidelberg, 2003. Springer-Verlag.

[76] Xuejia Lai, Rainer A Rueppel, and Jack Woollven. A Fast Cryptographic Checksum Algorithm Based on Stream Ciphers. In *Advances in Cryptology—AUSCRYPT'92*, pages 339–348. Springer, 1993.

[77] Joel Lathrop. Cube Attacks on Cryptographic Hash Functions. 2009.

[78] Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda AlKhzaimi, and Erik Zenner. A Cryptanalysis of PRINT Cipher: the Invariant Subspace Attack. In *Advances in Cryptology–CRYPTO 2011*, pages 206–221. Springer, 2011.

[79] Jian Liu and Lusheng Chen. On the Relationships between Perfect Nonlinear Functions and Universal Hash Families. *Theoretical Computer Science*, 513:85–95, 2013.

[80] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The Theory of Error Correcting Codes*, volume 16. Elsevier, 1977.

[81] Mitsuru Matsui. Linear Cryptanalysis Method for DES Cipher. In *Advances in Cryptology—EUROCRYPT'93*, pages 386–397. Springer, 1994.

[82] David A McGrew and John Viega. The Security and Performance of the Galois/Counter Mode (gcm) of Operation. In *Progress in Cryptology-INDOCRYPT 2004*, pages 343–355. Springer, 2005.

[83] Wilfried Meidl and Harald Niederreiter. On the Expected Value of the Linear Complexity and the K-Error Linear Complexity of Periodic Sequences. *Information Theory, IEEE Transactions on*, 48(11):2817–2825, 2002.

[84] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of Applied Cryptography*. CRC press, 1996.

[85] Ralph C Merkle. A Certified Digital Signature. In *Advances in Cryptology—CRYPTO'89 Proceedings*, pages 218–238. Springer, 1990.

[86] Ralph Charles Merkle. Secrecy, Authentication, and Public Key Systems. 1979.

[87] Paweł Morawiecki, Josef Pieprzyk, and Marian Srebrny. Rotational Cryptanalysis of Round-Reduced Keccak. In *Fast Software Encryption*, pages 241–262. Springer, 2014.

[88] Pawel Morawiecki, Josef Pieprzyk, Marian Srebrny, and Michal Straus. Preimage Attacks on the Round-Reduced Keccak with the Aid of Differential Cryptanalysis. Technical Report IACR 2013/561, International Association for Cryptologic Research, 2013.

[89] María Naya-Plasencia, Andrea Röck, and Willi Meier. Practical Analysis of Reduced-Round Keccak. In *Progress in Cryptology–INDOCRYPT 2011*, pages 236–254. Springer, 2011.

[90] Wim Nevelsteen and Bart Preneel. Software Performance of Universal Hash Functions. In *Advances in Cryptology—EUROCRYPT'99*, pages 24–41. Springer, 1999.

[91] NIST. The SHA-3 Competition (2007-2012).

[92] NIST. Recommendation for Block Cipher Mode of Operation: The CMAC Mode for Authentication. NIST Special Publication 800-38B. 2005.

[93] NIST. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. NIST Special Publication 800-38D. 2007.

[94] David Perez and Jose Pico. A Practical Attack Against GPRS/EDGE/UMTS/HSPA Mobile Data Communications. Presented in Black Hat Conference, 2011.

[95] Phillip Rogaway and Thomas Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In *Advances in Cryptology-EUROCRYPT 2006*, pages 373–390. Springer, 2006.

[96] Rainer A Rueppel. *Analysis and Design of Stream Ciphers*. Springer-Verlag New York, Inc., 1986.

[97] Markku-Juhani Olavi Saarinen. Cycling Attacks on GCM, GHASH and Other Polynomial MACs and Hashes. In *Fast Software Encryption*, pages 216–225. Springer, 2012.

[98] Victor Shoup. On Fast and Provably Secure Message Authentication Based on Universal Hashing. In *Advances in Cryptology—CRYPTO'96*, pages 313–328. Springer, 1996.

[99] Gustavus J Simmons. Authentication Theory/Coding Theory. In *Advances in Cryptology*, pages 411–431. Springer, 1985.

[100] Martin Stanek. On Cryptographic Properties of Random Boolean Functions. *Journal of Universal Computer Science*, 4(8):705–717, 1998.

[101] Douglas R. Stinson. The Combinatorics of Authentication and Secrecy Codes. *Journal of Cryptology*, 2(1):23–49, 1990.

[102] Douglas R Stinson. Combinatorial Characterizations of Authentication Codes. *Designs, Codes and Cryptography*, 2(2):175–187, 1992.

[103] Douglas R. Stinson. Universal Hashing and Authentication Codes. *Designs, Codes and Cryptography*, 4(3):369–380, 1994.

[104] Yin Tan, Kalikinkar Mandal, and Guang Gong. Characterization of Column Parity Kernel and Differential Cryptanalysis of Keecak, 2014.

[105] Iwata Tetsu and Kurosawa Kaoru. OMAC: One-Key CBC MAC. In *Pre-proceedings of Fast Software Encryption, FSE 2003*, FSE '03, pages 137–161. Springer-Verlag, 2002.

[106] Fuhr Thomas, Gilbert Henri, Reinhard Jean-Rene, and Marion Videau. A Forgery Attack on the Candidate LTE Integrity Algorithm 128-EIA3. Technical Report IACR 2010/168, International Association for Cryptologic Research, 2010.

[107] Henk CA Van Tilborg and Sushil Jajodia. *Encyclopedia of Cryptography and Security.* Springer Science & Business Media, 2011.

[108] Zhe-Xian Wan. Construction of Cartesian Authentication Codes from Unitary Geometry. *Designs, Codes and Cryptography*, 2(4):333–356, 1992.

[109] Mark N Wegman and J Lawrence Carter. New Hash Functions and Their Use in Authentication and Set Equality. *Journal of computer and system sciences*, 22(3):265–279, 1981.

[110] Teng Wu and Guang Gong. The Weakness of Integrity Protection for LTE. Technical Report CACR 2013-03, Centre For Applied Cryptographic Research, 2013.

[111] Teng Wu and Guang Gong. The Weakness of Integrity Protection for LTE. In *Proceedings of The Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 79–88. ACM, 2013.

[112] Aaron D Wyner. The Wire-tap Channel. *The Bell System Technical Journal*, 54(8):1355–1387, 1975.

[113] Bo Zhu, Yin Tan, and Guang Gong. Revisiting MAC Forgeries, Weak Keys and Provable Security of Galois/Counter Mode of Operation. In Michel Abdalla, Cristina Nita-Rotaru, and Ricardo Dahab, editors, *Cryptology and Network Security - 12th*

International Conference, CANS 2013, Paraty, Brazil, November 20-22. 2013. Proceedings, volume 8257 of *Lecture Notes in Computer Science*, pages 20–38. Springer, 2013.