The Weakness of Integrity Protection for LTE

Teng Wu and Guang Gong
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, ON N2L 3G1, Canada
{teng.wu, ggong}@uwaterloo.ca

Abstract

In this paper, we concentrate on the security issues of the integrity protection of LTE and present two different forgery attacks. For the first attack, referred to as a *linear forgery attack*, EIA1 and EIA3, two integrity protection algorithms of LTE, are insecure if the initial value (IV) can be repeated twice during the life cycle of an integrity key (IK). Because of the linearity of EIA1 and EIA3, given two valid Message Authentication Codes (MACs) our algorithm can forge up to 2^{32} valid MACs. Thus, the probability of finding a valid MAC is dramatically increased. Although the combination of IV and IK never repeats in the ordinary case, in our well-designed scenario, the attacker can make the same combination occur twice. The duplication provides the opportunity to conduct our linear forgery attack, which may harm the security of communication. To test our linear forgery attack algorithm, we generate two counter check messages and successfully forge the third one. We also examine the attack timing by simulating real communication. From the experimental results, our attack is applicable. The second attack is referred to as a *trace extension forgery attack*, which works only in theory. However, this attack is more general than the linear forgery attack. Known only one MAC and message pair, we can construct a different message, who has the same MAC as the original one, with the probability $\frac{1}{2^{16}}$. In this attack, trace function is applied to the message to shrink the guessing space.

Index Terms. Forgery, MAC, LTE, man-in-the-middle.

1 Introduction

After more than twenty years evolution, cellular system has evolved to the fourth generation. Security issues of the cellular system are attracting more and more attention, because the expenses of attacking the system are much cheaper than before.

Integrity protection can protect messages from being modified. It also can prevent the impersonation attacks. Thus, integrity protection is important in communication, especially in wireless channels. Compared with wired transmission, active eavesdropping in a wireless environment is relatively easy. Without integrity protection, attackers can modify messages transmitted over the air as they wish.

In the public key cryptography domain, integrity is protected by digital signatures; similarly, in symmetric key cryptography, it is protected by MACs. Because computation of digital signatures is inefficient, in some resource-constrained applications, integrity protection is always based on MACs.

A MAC-generating algorithm usually has two components: the underlying cipher and the upper-level structure. An underlying cipher could be the keyed hash function, block cipher or stream cipher. The input messages of the algorithm are allowed to have arbitrary length. Input messages pass through the underlying cipher and become cipher text. This cipher text is assembled by the upper-level structure to get a length-fixed string, which is the output of the MAC. There are many widely used MACs, such as HMAC [7], EMAC [11], XCBC [9], OMAC [18], TMAC [13] and XOR MAC [8].

One threat to integrity protection is forgery attacks. If an attacker wants to impersonate some identities or to modify some messages, he must have the ability to forge the MAC (or the signature in public key cryptography) of some specific data. In this paper, since the cellular communications are protected by MACs, we focus on a forgery attack on MAC. There are basically two ways to forge a MAC: breaking the underlying cipher or bypassing the underlying cipher. The former one is hard to achieve, since the underlying ciphers are usually chosen to be classical ciphers, which have already been proven to be secure in both theory and practice. Thus, attackers always choose the latter one, when they try to forge a MAC. We also chose the latter way, which means our goal is to bypass the underlying ciphers of the cellular system to forge a MAC.

To bypass the underlying cipher, the attacker can apply either a probabilistic method or a structure dependent method. The probabilistic method, for example, the birthday attack, can be launched at most MAC generating algorithms. Although its target is general, the result of a birthday attack is not very significant, because the searching complexity is still exponential. The attack is also not realistic, because in practice, the attacker is not allowed to make that many queries. In contrast, some structure-dependent attacks, which only target on certain MAC-generating algorithms, are more implementable. Our method being able to forge depends on the linear structures of some specific MACs. It is also a kind of structure-dependent method.

1.1 Applications of MAC in LTE

UMTS and LTE are two cellular standards proposed and maintained by 3GPP. UMTS is considered a third-generation cellular communication system. Its successor is called LTE, which is considered to be the fourth generation. As wireless communication systems, the resources of mobile devices are constrained. Thus, MACs are used to protect the integrity of these two systems. In UMTS, the underlying ciphers of MACs are Kasumi [3] and Snow 3G [1]. The former is a block cipher, and the latter is a stream cipher. When UMTS migrated to LTE, AES was standardized. Consequently, Kasumi has been replaced by AES. Besides AES and Snow 3G, a new stream cipher, ZUC [5], has been added into the current standard.

In UMTS, integrity protection algorithms are called UIAs. UIA1 [2] and UIA2 [4] are based on

Kasumi and Snow 3G respectively. In LTE, integrity protection algorithms are called EIA. EIA1 is adopted from UIA2, which is based on Snow 3G; EIA2 is based on AES, and EIA3 [4] is based on ZUC. We use EIA in this paper to name these integrity protection methods in accordance with LTE, the latest standard.

1.2 MAC Based On Stream Ciphers and EIA

Wireless communication is widely used in daily life. As a result, stream ciphers are becoming more important than ever before. Originally, most MACs were based on keyed hash functions and block ciphers. Now some researchers have turned to the design of MACs based on stream ciphers.

In the past, the research on stream ciphers based MACs has not received much attention as that of block ciphers or hash functions based MACs. Recently, researchers have realized the importance of MACs using stream ciphers. Some significant work has been presented, such as GMAC [15], Grain-128a [6], EIA1, and EIA3. The underlying cipher of GMAC is a block cipher in counter mode, which makes a block cipher act as a stream cipher. Thus, GMAC can work with other stream ciphers directly. Both EIA1 and EIA3 have adopted some ideas from GMAC with some modifications to form their own MAC-generating algorithms.

It is publicly acknowledged that EIA2 can be considered as a secure MAC-generating algorithm, because it utilizes AES as the underlying cipher and CMAC as the MAC structure. AES is widely used for commercial purposes, and gets many analyses from academic study. AES has proven to be secure so far both theoretically and practically. CMAC is a kind of CBC MAC. Such kinds of MACs have already received many analyses, and proven to be secure.

Unlike EIA2, grounds for trusting EIA1 and EIA3 are not so solid. Since they are not as popular as AES and CBC MAC, the security of these algorithms is still not well studied. To the authors' best knowledge, there has been no significant attack on EIA1 so far, which means it does not evolve after it is created. EIA3 receives more attacks. But only some of them are significant.

1.3 Related Work

There is already some work analyzing the EIA family. The following attacks are two of the most significant.

Cycling Attack [17] can be applied to all polynomial MACs. Since EIA1 is a kind of polynomial MAC, it also suffers such attacks.

ZUC and EIA3 were published later than Snow 3G and EIA1. However, EIA3 has received more analyses than EIA1. The attack [19] proposed by Thomas et al. makes EIA3 change from v1.3 to v1.5. After changing to v1.5, EIA3 does not suffer such attacks any more. However, it is still not immune to our new attack, which will be presented in Section 3.

1.4 Our Contributions

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. On top of such facts, we develop a method that, with known two valid MACs, we can forge up to 2^{32} valid MACs by introducing a coefficient λ. Since we have 2^{32} valid MAC-message pairs, the probability of finding a pair with valid MAC and valid message is quite high. 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 $\frac{1}{2^{32}}$. Now the probability that we get a meaningful MAC-message pair is increased to more than $\frac{1}{2^{32}}$. In fact, finding the λ that can generate a meaningful pair is much easier in practice than finding it in theory, because the messages usually have some specific structures. Those structures may shrink the searching space. In addition, such an attack does not aim at only EIA1 and EIA3 but also at some general polynomial MACs.

To prove that our linear forgery attack is doable, we create a scenario from which our attack can be launched. Such a scenario is based on the observation that the authentication of LTE and UMTS is not really mutual, although it is claimed to be a mutual authentication. In Extensible Authentication Protocol - Authentication and Key Agreement (EAP-AKA), only the server sends the 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 a server by checking the MAC of the authentication vector. Such a protocol leaves a hole for the replay attack. Ordinarily speaking, the replay attack cannot get anything. However, because of our attack, the replay attack makes the forgery possible.

The second attack that we propose in this paper is referred to as a trace extension forgery attack. For this attack, we can construct a new message, who has the same MAC as the known message, with the probability $\frac{1}{2^{16}}$. The observation is that we can extend the original message such that the difference between the original MAC and the new MAC is a function of β over $GF(2^{64})$, where $\beta \in GF(2^{16}) \subset GF(2^{64})$. Then the probability of guessing β is $\frac{1}{2^{16}}$, which is greater than the probability of guessing the MAC. If we get the correct β, then we can forge the message.

The rest part of this paper is organized as followings. In Section 2 we introduce MACs and the EIA family. In Section 3, we present some security issues of EIA1 and EIA3, and present the linear forgery attack. In Section 4, we show how the trace extension forgery attack works. Section 5 describes a scenario from which the linear forgery attack can be launched in practice, and shows some experimental results of our attack.

2 Preliminary

In this section, we introduce the definition of the linear operation. This concept is the vital part of our attack. Generally, most polynomial MACs, such as EIA1 and EIA3, have this linear property.

2.1 Notation, Definition and Data Representation

For the purpose of this paper, the *Linear Operation* is defined as

Definition 1 *$F(x)$ represents the operation F applied to x. If the operation F satisfies*

- $F(a+b) = F(a) + F(b)$
- $F(c \cdot a) = cF(a)$,

where a, b and c are from a Galois Field; a, b and $c \cdot a \in Domain\ of\ F$, then F is called linear operation.

We also need a function that can map an element in a Galois field to an integer.

Definition 2 *A function $Nat(x) : GF(2^{64}) \mapsto \mathbb{Z}$. $\forall x \in GF(2^{64})$, x can be represented as a binary number under the normal basis. Function $Nat(x)$ returns this binary number as an integer.*

Trace function maps an element in a Galois field to its subfield. We borrow the definition of trace function from Golomb and Gong [12].

Definition 3 *Let q be a prime or a power of a prime. For $\alpha \in F = GF(q^n)$ and $K = GF(q)$, the trace function $Tr_{F/K}(x)$, $x \in F$, is defined by*

$$Tr_{F/K}(x) = x + x^q + \cdots + x^{q^{n-1}},\ x \in F.$$

We can easily prove $Tr_{F/K}(x) \in GF(q)$. Thus we omit the proof [12] here.

Table 1 lists all notations used in this paper.

In the following part of this paper, all data variables are presented with the most significant bit on the left-hand side. For example, V is a 64-bit integer, $< V >= (V_0 V_1 \cdots V_{62} V_{63})_2$, where V_i is the i-th bit of V. V_0 is the most significant bit.

We use the term "package" and "message" to represent the protocol data unit and the information that carried by the package respectively. Package is composed of the header (used for control) and the message. Take the counter check message for example, it is a package of radio resource control (RRC) protocol. The control message sequence number (SQN) is the header, and the values of different counters are the messages. These two parts together are called a *package of the counter check message.*

Table 1: Notations used in this paper

Notations	Explainations
$a\|\|b$	Concatenation of a and b.
$(a)_s$	Represent a in a base-s form. Example: $(1001)_2$, $(1FA8D)_{16}$.
$< i >$	The binary representation of i. Example: $< (5)_{10} >= (101)_2$.
L^i	The left shift operator. L^1 is simplified as L. Example: $L(10011011)_2 = (00110110)_2$.
$+$	Bitwise exclusive or.
$\cdot$	Multiplication in finite field. Some times, without ambiguity, we directly write $a \cdot b$ as ab.
$[A]_{i..j}$	The i-th bit to the j-th bit of A.
$\mathbf{M}$	A vector. Example: the message composed of several blocks. $\mathbf{M}$=$M_0\|\|M_1\|\|\cdots\|\|M_{n-1}$.
MAC($\mathbf{M}$)	The MAC of the message $\mathbf{M}$.

2.2 CBC-MAC, XOR MAC and GMAC

In wireless communication network, MAC is one of the most important elements to secure the system. CBC-MAC and XOR MAC are two well-known MACs. CBC-MAC comes from the CBC mode of block ciphers. There are a lot of variations of CBC-MAC, such as EMAC [11], XCBC [9], OMAC [18] and TMAC [13]. XOR MAC [8] has two categories, XMACC and XMACR. Compared with CBC-MAC, the structure of XOR-MAC is relatively simple.

GMAC has different design compared with CBC-MAC and XOR MAC, because it is based on the counter mode of the block cipher. GMAC [15] is standardized by NIST in 2007. When it is used for encrypted authentication, it is called GCM. GCM outputs both the encrypted data and MAC. When it is only used to generate MAC without encryption, it is called GMAC. Unlike GCM, GMAC does not output the encrypted data. GMAC is composed of two parts, GCTR and GHASH. GCTR is the counter mode of a block cipher. GHASH is a polynomial hash function. Messages first are encrypted by GCTR, and then passed through GHASH.

2.3 EIA1, EIA2 and EIA3

Because both the underlying cipher and the upper-level structure of EIA2 are proven to be secure, we do not discuss it in this paper. Hence, in this section, we present more details of EIA1 and EIA3 than of EIA2.

2.3.1 EIA1

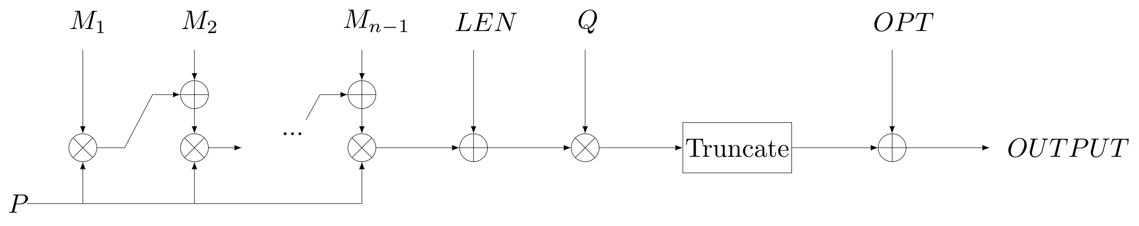

Figure 1: EIA1: based on Snow 3G

EIA1 is adopted from UIA2. The underlying cipher is Snow 3G. In the evaluation of EIA1, it is said that EIA1 comes from GMAC. In fact, it only borrows the idea of GHASH. Thus, EIA1 is considered to be a polynomial MAC. Details of EIA1 are shown in Figure 1. At the beginning stage of EIA1, Snow 3G generates a 160-bit key stream. Then this key stream is truncated into P, Q and OPT. P and Q

are 64-bit words. OPT is a 32-bit word. In Figure 1, M_i is the i-th block of message. The length of each block is 64 bits.

The mathematical description of EIA1 is given by

$$MAC(\mathbf{M}) = \left[\left(\sum_{i=1}^{n} P^i \cdot M_{n-i} + LENGTH\right) \cdot Q\right]_{0..31} + OPT. \tag{1}$$

Equation 1 can be written as

$$\begin{aligned} MAC(\mathbf{M}) &= \underbrace{\left[\sum_{i=1}^{n} P^i \cdot M_{n-i} \cdot Q\right]_{0..31}}_{part1} + \\ & \underbrace{[LENGTH \cdot Q]_{0..31} + OPT}_{part2}. \end{aligned}$$

Part 1 is a linear operation of the message. Part 2 is a value only relevant to the length of the message. If the length is fixed, part 2 is a constant. So MAC is computed by a linear combination of one linear block operation and a constant.

2.3.2 EIA2

Kasumi is replaced by AES in the new standard. Accordingly, EIA2 is put into the standard to replace UIA1. The underlying cipher of EIA2 is AES. As a block cipher, AES can use some existing MAC generating algorithms without any changes. CMAC [14] is chosen as the MAC generating algorithm of EIA2. CMAC is a kind of CBC-MAC, which has already been well studied.

2.3.3 EIA3

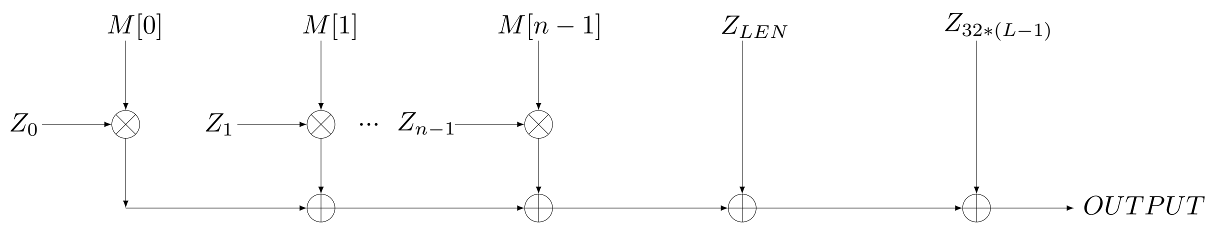

Figure 2: EIA3: based on ZUC

Originally, ZUC is not in UMTS. It is added to the standard after the system migrates to LTE. The integrity protection based on ZUC is EIA3. The same as EIA1, EIA3 is also claimed to be GMAC. However, Figure 2 suggests that it is much closer to XOR MAC. In Figure 2, $M[i]$ is the i-th bit of message, Z_i is the i-th word of the key stream generated by ZUC. Z_i starts with the i-th bit of the key stream. 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 is given by

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

where $M[i]$ is the i-th bit of message; z_i is the i-th word in the key stream, i.e. $z_i = z[i]||z[i+1]|| \cdots ||z[i+31]$; $L = \lceil LENGTH/32 \rceil + 2$. The main observation of EIA3 is that Part 1 is a linear operation. This is very straightforward:

$$\begin{aligned} \sum_i (M[i] + M'[i]) z_i &= \sum_i (M[i] z_i + M'[i] z_i) \\ &= \sum_i M[i] z_i + \sum_i M'[i] z_i. \end{aligned}$$

Part 2 is a constant if the length is fixed.

2.4 IV Synchronization Mechanism

EIA1, EIA2 and EIA3 synchronize IVs in the same way. As required by the EIA family, $COUNT-I$ and $FRESH$ are input as parameters to form IV of the underlying cipher. $FRESH$ is a random number. $COUNT-I$ is a counter that records how many times IK has already been used so far. It contains two parts, SQN and hyper frame number (HFN). SQN is the sequence number. It is increased after a package is sent. When SQN overflows, HFN is increased. $COUNT-I$ and $FRESH$ together are used to prevent replay attacks.

$FRESH$ and $COUNT-I$ are written in the package to synchronize the transmitter and receiver, which maintain two counters, named $COUNTER_{tx}$ and $COUNTER_{rx}$, respectively. The transmitter uses $COUNTER_{tx}$ as the value of its $COUNT-I$, and then generates key streams. 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 $COUNTER_{rx}$, $FRESH$ is used to form IV together with $COUNT-I$. Then the key streams generated by this IV and IK are used to verify the MAC. If the value of $COUNT-I$ is smaller than $COUNTER_{rx}$, this package will be disregarded.

3 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 MACs to forge another valid MAC.

3.1 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 $GF(2^{64})$, 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 $GF(2^{64})$, and let $\beta = \alpha^{2^{32}+1}$. Then β is a primitive element of $GF(2^{32})$, a subfield of $GF(2^{64})$. The minimal polynomial of α over $GF(2^{32})$ is given by

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

where $u = \beta^{17}$, $v = \beta$. Then each element in $GF(2^{64})$ can be represented as $a + b\alpha$, $a, b \in GF(2^{32})$. We define

$$f(\mathbf{M}, P) = \sum_{i=1}^{n} M_{n-i}P^i, for\ P \in GF(2^{64}),\ M_i \in GF(2^{64}).$$

Property 1 *For any* $\lambda \in GF(2^{32}) \subset GF(2^{64})$,

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

then

$$f(\boldsymbol{M}_1 + \boldsymbol{M}_2, P) = f(\boldsymbol{M}_1, P) + f(\boldsymbol{M}_2, P) \quad (2)$$
$$f(\lambda \boldsymbol{M}, P) = \lambda f(\boldsymbol{M}, P). \quad (3)$$

Proof 1 *According to the definition of* $f(\boldsymbol{M}, P)$*, we have*

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

Thus, the assertions are true.

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

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

Let $Q \cdot f(\mathbf{M}, P) = a + b\alpha$, $Length \cdot Q = c + d\alpha$, a, b, c, and $d \in GF(2^{32})$. From the MAC generation of EIA1 in Section 2.3.1 and Property 1, the following result follows immediately.

Property 2 *For any* $\lambda \in GF(2^{32})$,

$$\begin{aligned} MAC(\mathbf{M}) &= a + c + OPT & (4)\\ MAC(\lambda\mathbf{M}) &= \lambda a + c + OPT. & (5)\end{aligned}$$

EIA3 has the same property of Property 2. In other words, 4 and 5 are true, where MAC is generated by EIA3. The proof is straightforward. Thus, we omit it here.

3.2 Linear Forgery Attack Algorithm

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

$$Q \cdot f(\mathbf{M}_i, P) = a_i + b_i\alpha,\ a_i,\ b_i \in GF(2^{32}). \tag{6}$$

Theorem 1 *Let* (i, j, k) *be a permutation of* $(1, 2, 3)$. *For any* $\lambda \in GF(2^{32})$

$$MAC(\mathbf{M}_{new}) = \lambda(MAC(\mathbf{M}_i) + MAC(\mathbf{M}_j)) + MAC(\mathbf{M}_k) \tag{7}$$

which is a valid MAC value of the message

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

Proof 2 *We give a proof only for* $(i, j, k) = (1, 2, 3)$, *since the proofs for the other cases are similar. In order to prove (7), we compute the results of both sides of (7). According to Properties 1 and 2*

$$MAC(\boldsymbol{M}_{new}) = \lambda(a_1 + a_2) + a_3 + c + OPT. \tag{8}$$

On the other hand,

$$\begin{aligned} & \lambda(MAC(\boldsymbol{M}_1) + MAC(\boldsymbol{M}_2))) + MAC(\boldsymbol{M}_3) \\ = \;& \lambda(a_1 + c + OPT + a_2 + c + OPT) + a_3 + c + OPT \\ = \;& \lambda(a_1 + a_2) + a_3 + c + OPT. \qquad (9)\end{aligned}$$

The assertion follows from (8) and (9).

From (7), we have the following corollary.

Corollary 1 *Let (i, j) be a permutation of $(1, 2)$, and $k \in (1, 2)$. For any $\lambda \in GF(2^{32})$, (7) is true. In other words, if we have the valid MACs from two queries, then*

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

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 to forge a valid MAC by using two known valid MACs is shown in Algorithm 1, where $find\lambda()$ is a function that returns a λ such that either $\lambda(\mathbf{M}_1 + \mathbf{M}_2) + MAC(\mathbf{M}_1)$ or $\lambda(\mathbf{M}_1 + \mathbf{M}_2) + MAC(\mathbf{M}_2)$ is a valid message. How to find λ, such that the message is also valid, is discussed in Section 3.3.

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 *EIA's MAC has 32 bits. An attacker wishes to forge a valid MAC, it is equivalent to him randomly selecting 32 bits; the probability of success is $\frac{1}{2^{32}}$. However, if the attacker can make two* *queries for obtaining two valid MACs, then he can forge 2^{32} messages with valid MACs. In Section 5, we will demonstrate how the attacker can obtain two valid MACs in practice.*

3.3 How To Find λ

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

Usually in a real environment, it is easier to find λ, because there is a relationship between these two known messages, such as the relationship between two counter check messages. Two counter check messages have very similar structures. Therefore, most bits in the exclusive or of two counter check messages are zeros. We need to consider only very few bits, which are nonzero.

Moreover, even if we cannot find the valid message, our linear forgery attack can still cause some 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, the computational resource will be occupied by verifying and decoding.

4 Trace Extension Forgery Attack

This attack focuses only on EIA1. Assume, we have the message $\mathbf{M} = M_0||M_1||\cdots||M_{n-1}$ and $MAC(\mathbf{M})$. From EIA1 we know

$$MAC(\mathbf{M}) = \left[\left(\sum_{i=1}^{n} P^i \cdot M_{n-i} + LENGTH\right) \cdot Q\right]_{0..31} + OPT.$$

Let

$$g(\mathbf{M}) = \sum_{i=1}^{n} P^i \cdot M_{n-i}.$$

Then we have the Lemma 1.

Lemma 1 *Extend the message* $\boldsymbol{M}$ *to* $\boldsymbol{M}_{new} = c_0||\cdots||c_{N-n-1}||c_{N-n} + M_0||\cdots||c_{N-1} + M_{n-1}$*, where* $N \geq 2^{48}$*,*

$$c_i = \begin{cases} \alpha, & \text{for } i = N-1,\ N-2^{16},\ N-2^{32}, N-2^{48},\ \alpha \in GF(2^{64}); \\ 0, & \text{others.} \end{cases}$$

Let $F = GF(2^{64})$ *and* $K = GF(2^{16})$*. Then we have*

$$g(\boldsymbol{M}_{new}) = g(\boldsymbol{M}) + \alpha \cdot Tr_{F/K}(P).$$

Proof 3 *The proof is very straightforward.*

$$\begin{aligned} g(\boldsymbol{M}_{new}) &= \sum_{i=1}^{n} P^i(M_{n-i} + c_{N-i}) + \sum_{i=n+1}^{N} P^i c_{N-i} \\ &= g(\boldsymbol{M}) + \alpha \cdot (P^{2^{48}} + P^{2^{32}} + P^{2^{16}} + P) \\ &= g(\boldsymbol{M}) + \alpha \cdot Tr_{F/K}(P). \end{aligned}$$

Theorem 2 *If the integrity protection method is EIA1, then the probability of finding a message, which has the same MAC as a known message, is no less than* $\frac{1}{2^{16}}$.

Proof 4 *Assume we have a message* $\boldsymbol{M}$*, whose length is* LEN*, and its MAC is* $MAC(\boldsymbol{M})$*. We construct a method that can find another message, whose MAC is also* $MAC(\boldsymbol{M})$*. From Lemma 1,* $\boldsymbol{M}_{new}$ *has the property that* $g(\boldsymbol{M}_{new}) = g(\boldsymbol{M}) + \alpha \cdot Tr_{F/K}(P)$*. Since* α *can be arbitrary element in* $GF(2^{64})$*, thus we always have at least one* α *such that* $Nat(\alpha \cdot Tr_{F/K}(P) + LEN) \geq Nat(LEN)$*, where* $Nat(x) : GF(2^{64}) \mapsto \mathbb{Z}$ *is the function we defined before. Let* $t = Nat(\alpha \cdot Tr_{F/K}(P) + LEN) - Nat(LEN)$*. We pad* t *zeros to the left side of* $\boldsymbol{M}_{new}$ *to get another message denoted by* $\boldsymbol{M}_f$*. Obviously,* $g(\boldsymbol{M}_{new}) = g(\boldsymbol{M}_f)$*. Thus the MAC of* $\boldsymbol{M}_f$ *is*

$$\begin{aligned} MAC(\boldsymbol{M}_f) &= \left[(g(\boldsymbol{M}_{new}) + \alpha \cdot Tr_{F/K}(P) + LEN) \cdot Q\right]_{0..31} + OPT \\ &= \left[(g(\boldsymbol{M}) + \alpha \cdot Tr_{F/K}(P) + \alpha \cdot Tr_{F/K}(P) + LEN) \cdot Q\right]_{0..31} + OPT \\ &= [(g(\boldsymbol{M}) + LEN) \cdot Q]_{0..31} + OPT \\ &= MAC(\boldsymbol{M}). \end{aligned}$$

To get $\boldsymbol{M}_f$*, we need to guess* $Tr_{F/K}(P)$*. Since* $Tr_{F/K}(P)$ *maps* P *to* $GF(2^{16})$*, the probability of guessing* $Tr_{F/K}(P)$ *is* $\frac{1}{2^{16}}$.

Remark 2 *The proof of Theorem 2 naturally gives a way to find the message, who has the same MAC as the original one. Instead of guessing the MAC, whose probability is* $\frac{1}{2^{32}}$*, our method tries to find the message, who has the specific MAC. The success probability of trace extension forgery attack is* $\frac{1}{2^{16}}$*, which is much greater than* $\frac{1}{2^{32}}$.

5 Application of Linear Forgery Attack

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

5.1 A Scenario of Fixing IV

Figure 3 demonstrates the procedure by which we can get the same SQN together with the same IK. The preconditions 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. [16] show that Condition (1) is applicable. Condition (2) is also easy to satisfy. We can choose the Android smart phone to be our target because Android is an open platform, which is popular around the world.

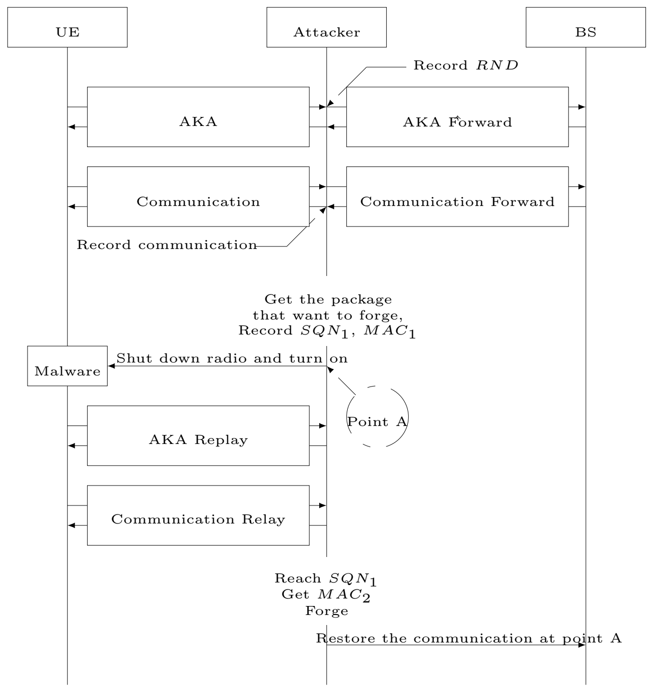

Figure 3: Fixing IV

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. Thus, the victim believes it is talking with the real base station. Notice that the EAP-AKA is claimed to be mutually authenticated. The user equipment (UE) proves its identity to an radio network controller (RNC) by replying to the challenge from the RNC. However, since the UE does not send the challenge to the RNC, the RNC can prove itself only by computing 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 RNC. 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 SQN. As long as we get two identical SQNs with the same IK, we can launch our linear forgery attack.

After 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 base station, 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 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.

5.2 Counter Check Message

A realistic application of our linear forgery attack is that we can forge the counter check message. In LTE, the integrity of the user plane is not protected. Thus, the counter check message is sent from the RNC to the UE to check the number of data packages that have been transmitted. The RNC includes the most significant s bits of its counter in the counter check message. When the UE receives the counter check message, it compares its own counter with the value included in the counter check message and sends its counter's value back. If the difference is not acceptable, the RNC will release the connection. This procedure is shown in Figure 4. Chen et al. [10] present more details about the counter check message.

We want to forge the counter check message because sometimes the attacker inserts some data into user plane data. If the counter check message is conducted correctly, the RNC will find out the insertion. For example, the MITM attacker inserts a redirect URL command or advertisement into the web page that the user is browsing. He must expect that RNC cannot detect the insertion. Then he needs to modify the counter check message.

5.3 Launching Attack

We assume that the MAC-I in Figure 4 is generated by EIA1 or EIA3. IVs of EIA1 and EIA3 are composed of two portions. The least significant four bits represent the Radio Resource Controller sequence number (RRC SQN). The other twenty-eight bits represent the HFN. Each time an RRC

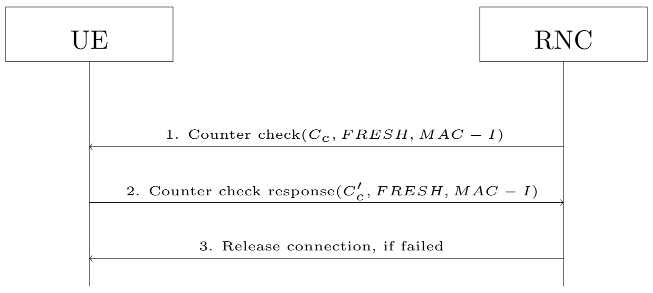
UE
RNC
1. Counter check(C_c, $FRESH$, $MAC-I$)
2. Counter check response(C'_c, $FRESH$, $MAC-I$)
3. Release connection, if failed

Figure 4: Counter Check Message.

signal is sent, the RRC SQN is increased by one. If there is an overflow of the RRC SQN, the HFN is increased by one.

Attacking scenario:

1. RNC 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 RNC finds the difference is acceptable, and continues to communicate with the MITM attacker.
6. The MITM attacker can continue to forward messages between the RNC and the UE without being detected by the RNC.

This process creates a forged MAC. In this procedure, there is a drawback, i.e., the connection between the MITM attacker and the RNC may 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.

5.4 Experimental Results

In order to test our attack, we generate two counter check messages, in which there are two counters, as shown in Table 2. The RRC commands of these two packages are listed in Table 3.

Table 2: 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: 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

5.4.1 Forge Procedure

The xor sum of these two messages is

```
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 $\mathbf{M}_{new} = \mathbf{M}_2 + \lambda(\mathbf{M}_1 + \mathbf{M}_2)$

```
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 message is represented in a binary form. Decoding the binary message, we get the result shown in Figure 5, which demonstrates the xml form of the binary message. From Figure 5, it is obvious that the forged message still contains the counter values of the bearers 10 and 50. The uplink and downlink counter values of the bearer 10 are 577 and 531 respectively. They are increased compared with the real values. The counter values of the bearer 50 are increased as well. The forged value of uplink counter is 274, and the forged downlink counter value is 352. We can also verify the MAC of the message $\mathbf{M}_{new}$. $MAC(\mathbf{M}_{new})$ is indeed $MAC_2 + \lambda(MAC_1 + MAC_2)$. Therefore we successfully forge a valid package, in which the value of each counter is increased, and the MAC of the message is valid. This means if the attacker inserts some packages, RNC will not realize that.

```
<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 5: Decoding result

5.4.2 Timing of Attack

Turning the radio off and on usually costs three to six seconds. If RNC does not time out within this range, our attack can be launched. There is no requirement for this time-out duration in the standard. It is decided by the manufacturer. We cannot make a direct test, because analysis of public communications is forbidden by the law. However, since RRC commands are sent in an ARQ fashion, we can use the retransmitting time to show the time out duration intuitively. We simulate an Additive White Gaussian Noise (AWGN) channel. The modulation scheme is Quaternary Phase Shift Keying (QPSK). The result is shown in Figure 6. 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 6 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 that message. If users are in a building, such a situation may occur with high probability. To make sure all users can get services, the time out duration must be relatively long. This may give us a chance to conduct the attack.

6 Conclusion and Future Work

In this paper, we proposed a method whereby two known valid MAC-message pairs generated by the same IV and IK, we can forge 2^{32} valid MAC-message pairs. In a real environment, we can easily find a meaningful MAC-message pair among those 2^{32} pairs. We also developed an attack that makes

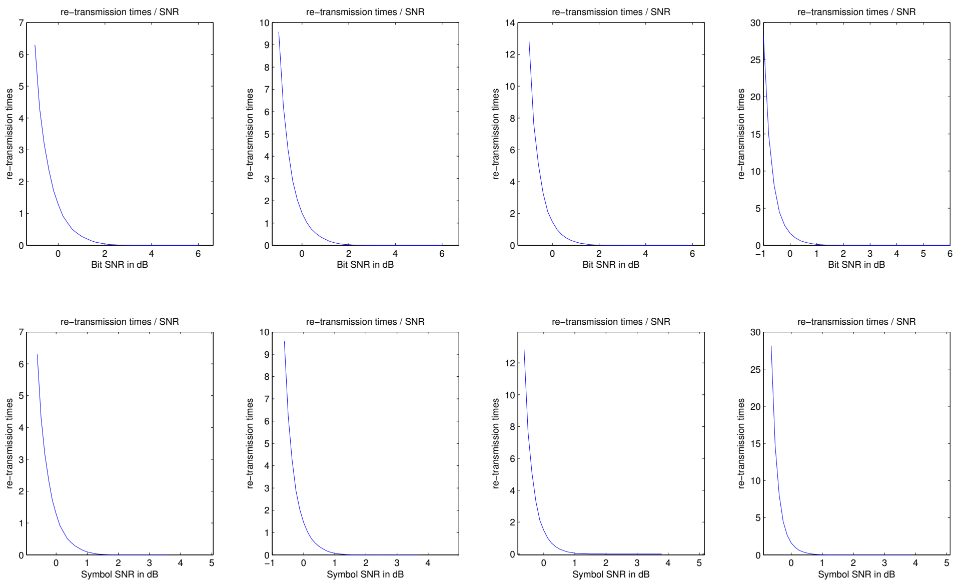

Figure 6: Retransmission times / SNR

the same IV and IK occur twice. This 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. The problem is how to find a way that can avoid linear structures without compromising efficiency. So far, this issue seems to be a trade-off.

Although the trace extension forgery attack is only theoretical, the success probability $\frac{1}{2^{16}}$ is considerable. Besides, it needs merely one pair of message and MAC. This attack shows the polynomial like MAC is not secure.

In the next stage of our research, the way to fix IV needs some improvement, for it heavily depends on the timing. If we can find a better way to get the same IV and IK twice, then our attack can launch at any time of communication (not only the time not too long after powering up), and also the failure rate caused by timing out will be reduced.

The trace extension attack can only construct the message, whose length is more than 2^{48} blocks. This number is too huge to be realistic. To improve, we need to reduce the length of the message. Moreover, if we can map P to a subfield, which is smaller than $GF(2^{16})$, then the probability will be higher than $\frac{1}{2^{16}}$.

References

[1] 3GPP. Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2. Document 2: SNOW 3G Specification. September 2006.

[2] 3GPP. Specification of the 3GPP Confidentiality and Integrity Algorithms. Document 1: f8 and f9 Specification. 2007.

[3] 3GPP. Specification of the 3GPP Confidentiality and Integrity Algorithms. Document 2: Kasumi Specification. June 2007.

[4] 3GPP. Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3. Document 1: 128-EEA3 and 128-EIA3 Specification. January 2011.

[5] 3GPP. Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3. Document 2: ZUC Specification. January 2011.

[6] M. Agren, M. Hell, T. Johansson, and W. Meier. Grain-128a: A New Version of Grain-128 with Optional Authentication. *Int. J. Wire. Mob. Comput.*, 5(1):58–59, December 2011.

[7] M. Bellare, R. Canetti, and H. 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.

[8] M. Bellare, R. Guérin, and P. 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.

[9] J. Black and P. 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.

[10] L. Chen and G. Gong. *Communication System Security*, chapter 10, pages 358–359. CRC Press Taylor & Francis Group, 2012.

[11] P. Erez and R. Charles. CBC MAC for Real-Time Data Sources. *J. Cryptology*, 13:315–338, 2000.

[12] S. W. Golomb and G. Gong. *Signal Design for Good Correlation For Wireless Communication, Crypography, and Radar.* Cambridge University Press, 2005.

[13] K. Kurosawa and T. 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.

[14] NIST. Recommendation for Block Cipher Mode of Operation: The CMAC Mode for Authentication. NIST Special Publication 800-38B. 2005.

[15] NIST. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. NIST Special Publication 800-38D. 2007.

[16] D. Perez and J. Pico. A Practical Attack Against GPRS/EDGE/UMTS/HSPA Mobile Data Communications. Presented in Black Hat Conference, 2011.

[17] M.-J. O. Saarine. Cycling Attacks on GCM, GHASH and Other Polynomial MACs and Hashes. In *Proceedings of the Fast Software Encryption 2012*, 2012.

[18] I. Tetsu and K. Kaoru. OMAC: One-Key CBC MAC. In *Pre-proceedings of Fast Software Encryption, FSE 2003*, FSE '03, pages 137–161. Springer-Verlag, 2002.

[19] F. Thomas, G. Henri, R. Jean-Rene, and M. Videau. A Forgery Attack on the Candidate LTE Integrity Algorithm 128-EIA3. *IACR Cryptology ePrint Archive*, 2010:168, 2010.