Security Assessment of TUAK Algorithm Set

PROJECT REPORT

by

Guang Gong, Kalikinkar Mandal, Yin Tan, Teng Wu

{ ggong, kmandal, yin.tan, teng.wu }@uwaterloo.ca

Communications Security Lab
Department of Electrical and Computer Engineering
University of Waterloo, Canada

October 30, 2014

Contents

List of Tables

List of Figures

Abbreviations

3GPP	3rd Generation Partnership Project
AES	Advanced Encryption Standard
AMF	Authentication Management Field
AK	Anonymity Key
AuC	Authentication Centre
CBC	Cipher Block Chaining
CK	Cipher Key
IK	Integrity Key
IN	A 1600-bit value that is used in f_1 to f_5^* algorithms
INSTANCE	an 8-bit value that is used to specify different modes of operation and different parameter lengths within the algorithm set.
IV	Initialisation Vector
K	Subscriber Key
MAC	Message Authentication Code
MAC-A	Network Authentication Code
MAC-S	Resynchronisation Authentication Code
OUT	A 1600-bit Output of the Keccak Permutation
RES	Response to Challenge
RNC	Radio Network Controller
SAGE	Security Algorithms Group of Experts
SQN	Sequence Number
UE	User Equipment
UMTS	Universal Mobile Telecommunications System
USIM	User Services Identity Module
XRAM	Extended RAM
XRES	Expected User Response

Notations

$\mathbb{F}(p^n)$	The finite field of p^n elements, where p is a prime number
$\mathbb{F}_p^n$	Vector space of dimension n over $\mathbb{F}_p$
Π	The Keccak Permutation
f, g	Boolean functions over $\mathbb{F}_{2^n}$
F, G	Vectorial Boolean functions from $\mathbb{F}_{2^n}$ to $\mathbb{F}_{2^m}$
$d(f)$	the algebraic degree of a Boolean function f
$\mathrm{NL}(f)$	the nonlinearity of a Boolean function
$\mathrm{LC}(\mathbf{s})$	the linear complexity of a binary sequence $\mathbf{s}$ with period N
$\lvert X\rvert$	the length of a binary word X
$\lvert S\rvert$	the cardinality of a set S

Abstract

The 3^{rd} Generation Partnership Project (3GPP) aims at the development of the specifications for the next generation cellular system. The TUAK algorithm set, which contains authentication and key generation algorithms, is proposed as an alternative to MILENAGE for the 3GPP. The design of algorithms in TUAK is based upon the winner of the SHA-3 competition, Keccak's permutation Keccak-$f[1600]$. It contains eight different algorithms, namely $\mathrm{TOP}_C, f_1, f_1^*, f_2, f_3, f_4, f_5, f_5^*$. The f_1 (and f_1^* that is used as re-synchronisation message authentication) algorithm ensures the authenticity of messages, f_2 is used for generating responses and f_3 to f_5 (f_5^*) are used as key derivation functions.

In this report, we provide a comprehensive security appraisement of the TUAK algorithm set. In summary, our security assessment on TUAK consists of three phases. First, we analyze the security of TUAK algorithms by performing various relevant cryptanalytic attacks on authentication and key derivation functions. We show the attack resistant properties of the TUAK algorithm set by providing the complexity of the attacks. Second, we present the security proofs on the soundness of the algorithms in TUAK when they are used as message authentication codes and key derivation algorithms. Finally, to further explore the security properties of TUAK, we develop a new cryptanalytic method, called *multi-output filtering model*, and apply it on TUAK. We study the algorithm f_1 in more detail since it is the MAC generating function and more suitable for our attacking scenario. To better measure the performance of TUAK algorithms under this new attack, we also apply this technique on AES, KASUMI and PRESENT. Our study shows that f_1 has very good randomness properties and performs similarly to AES under the same model.

Chapter 1

Introduction

The 3^{rd} Generation Partnership Project (3GPP) aims at the development of the specifications for the next generation cellular system. Security and privacy of the cellular system is an important aspect of the specifications. In the specification numbering system of the 3GPP, the series 33 and 35 are related to the security aspects of the cellular system. The series 33 is concerned about the general security infrastructures and architectures, and the series 35 describes the security algorithms. In the document numbered as TS 35.205, the 3GPP defines a set of functions called MILENAGE. This function set contains eight functions TOP_C, f_1, f_1^*, f_2, f_3, f_4, f_5, and f_5^*. MILENAGE is used in the authentication and key agreement routine of the cellular network and TUAK is designed as an alternative set of authentication and key derivation algorithms.

For the security consideration, a protocol should have a cipher suite instead of only one choice. Thus, a new function set that can serve the same as MILENAGE is necessary for the cellular system. Recently, NIST finalized the selection of the SHA-3 algorithm and announced Keccak as the winner of this competition due to its novel design and excellent performance both in resistance to various attacks and implementation in software and hardware (one may refer to NIST for details about this selection process). The structure of Keccak is based on the so-called sponge functions, which are extensively studied by the community. As described by the designer of the new algorithm set under evaluation in this report, proposing a new set of functions based on Keccak becomes the most efficient way to enhance the security of the cellular network. TUAK is such a new algorithm set. It directly applies Keccak to construct the functions TOP_C and f_1 to f_5. Although Keccak is fully studied by the researchers, as an application of Keccak, TUAK still needs to be well analyzed to avoid inappropriately apply Keccak such that the security properties of Keccak are decreased. TUAK serves as the authentication and the key derivation functions, which are the essential functions of an authentication and key agreement protocol. Therefore, an analysis of the construction of the authentication and key derivation functions is crucial.

1.1 Background

A series of functions are used in the authentication procedure of cellular network. Formally, these functions are named as f_i, where $1 \le i \le 5$. To fully understand these functions, we first briefly describe the authentication procedure. After that, the role of each f_i and the parameters of each f_i are presented in the second part of this section.

1.1.1 The EPS-AKA

The *EPS-AKA* is the authentication and key agreement protocol of the 4G LTE network. It is a sequence number based mutual authentication protocol. The basic framework is the challenge-response authentication. The purpose of the whole authentication process is to prove that each party has the same long term credential K and to derive the master session key based on K. Notice that K resides in the subscriber identity module (SIM) and the Authentication Centre (AuC) only. It cannot be moved or copied. The whole process is shown in Figure 1.1.

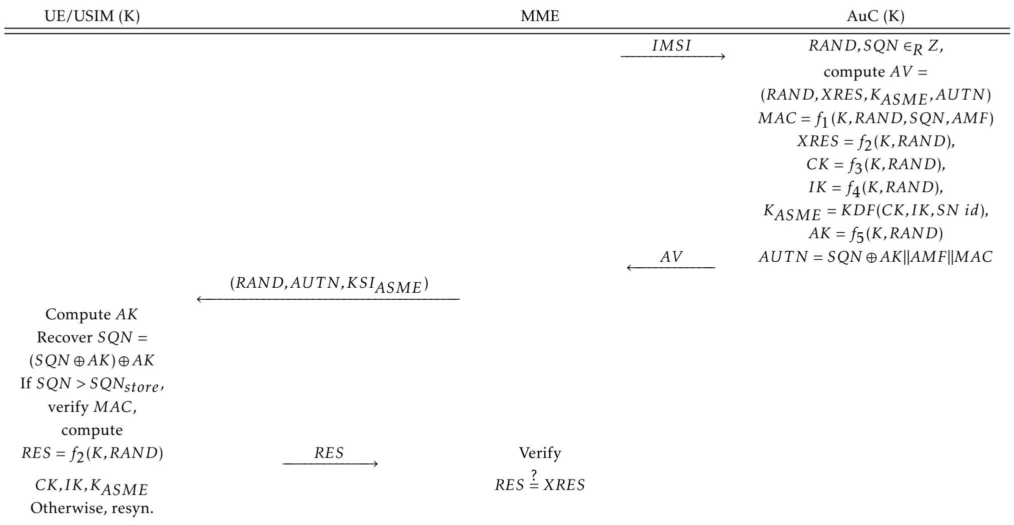

Figure 1.1: The EPS-AKA Procedure

When the user equipment (UE) is powered on, the UE sends the international mobile subscriber identity (IMSI) to the mobility management entity (MME). The MME forwards this IMSI to the AuC. AuC fetches the long term credential K to generate a batch of authentication vectors (AV). Each AV_i is composed as

$$AV_i = (RAND_i, XRES_i, K_{ASME_i}, AUTN_i).$$

$RAND_i$ is a random number; $XRES_i$ is the expected response value; K_{ASME_i} is the session key derived by the random number, sequence number and other parameters; $AUTN_i$ is the authentication token, which contains several fields as shown below.

$$AUTN_i = SQN_i \oplus AK_i \| AMF_i \| MAC_i.$$

SQN_i is the sequence number; AK_i is the anonymity key; AMF_i is the authentication management field; MAC_i is the message authentication code. The AuC sends all AVs to the MME. By receiving several AVs from the AuC, the MME may conduct the local authentication without knowing the long term credential or involving the AuC. This mechanism is useful especially for the roaming case. Considering the case that the user is roaming out of her country, if each authentication must involve the AuC, the cost of the authentication would be increased dramatically.

The MME retrieves the random number and the authentication token and sends to the UE. In Figure 1.1, the MME also sends the KSI_{ASME} to the UE. The KSI_{ASME} is like the handle that refers to the K_{ASME_i}. After the UE gets the packet sent by the MME, it first checks the freshness of the SQN_i by comparing with a stored value. Then the UE computes the MAC_i' of the $AUTN_i$ and compares the MAC_i' with the MAC_i. Notice that the MAC_i does not only protect the integrity of the $AUTN_i$, but also let the UE authenticate the network. Only when the UE knows the network is real for sure, it computes the response and sends the response back to the MME. By checking the response with the expected response, the MME can authenticate the UE.

1.1.2 Function f_i and Parameters

Before TUAK, there is only one set of functions called MILENAGE to server as the functions f_1 to f_5. MILENAGE is illustrated in Figure 1.2. The c_i and r_i, for $1 \leq i \leq 5$, are constants. OP_c is a operator specified constant. E_k is any block cipher with block length 128 bits. Usually, the network operators prefer to chose 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 is called the user authentication function. It generates the expected response and response. The f_3 and f_4 functions generate the CK and IK respectively. The CK and IK are legacies of the 3G UMTS network. They are the key 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

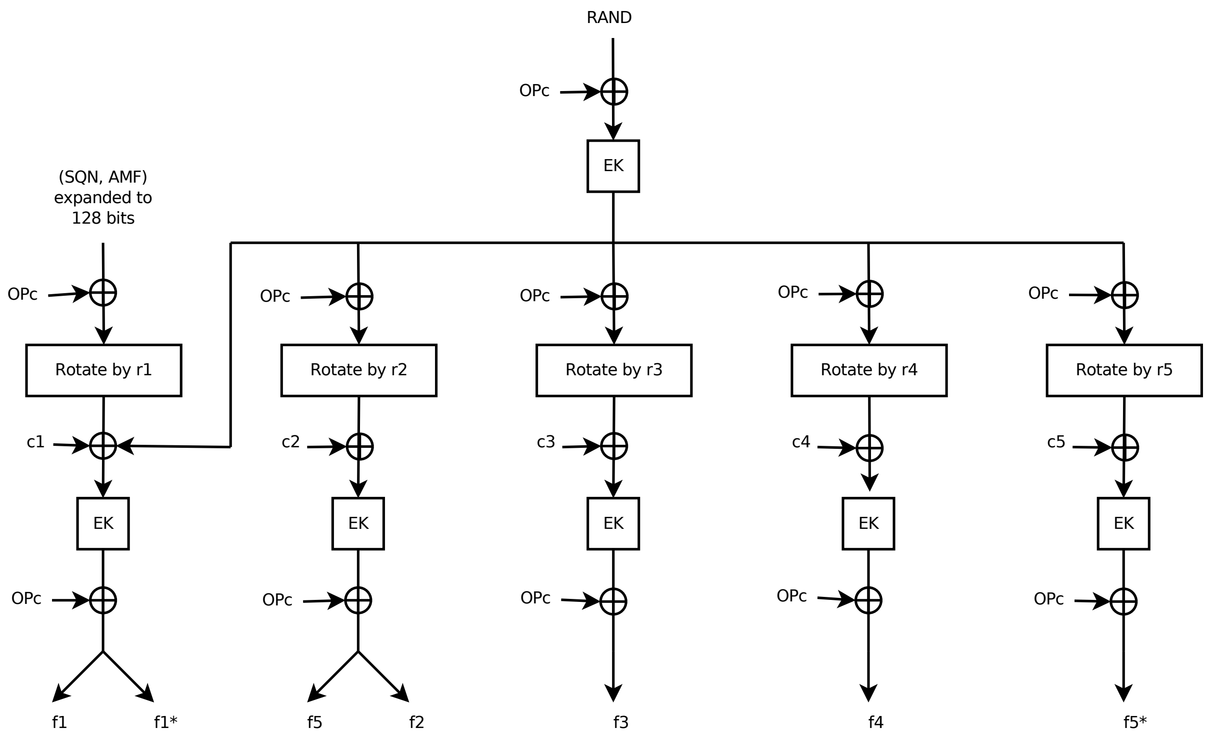

Figure 1.2: MILENAGE

function f_5. Notice that the SQN may mismatch, say smaller than the stored value. When such situation happens, the UE and the AuC run a re-synchronization routine. For the security reason, the MAC and AK in re-synchronization are generated by f_1^* and f_5^* respectively.

1.2 Our contribution

In this report, we provide a comprehensive cryptanalysis of the algorithms in TUAK. We start from showing the resistance of the algorithms to various classical attacks by presenting the complexity of these attacks on them. For those attacks already applied on Keccak by other researchers, we discuss their influence on TUAK in Chapter 3; and for those not applying directly on Keccak but are applicable on key-mode Keccak, namely suitable for the algorithms in TUAK, we discuss their impact on TUAK in Chapter 4. Our results show that all algorithms in TUAK inherit the security properties from Keccak perfectly.

Another important contribution in this report is that we provide the security proofs on the soundness of algorithms in TUAK when they are used as MAC and key generating algorithms. This work further guarantees the security of TUAK algorithms. One may refer to Chapter 5 for the security proofs.

Finally but importantly, we develop a new type of cryptanalytic technique, called *multi-output filtering model*. This technique is a generalization of the classical filtering model, which is widely applied on studying the security of stream ciphers. The difference between these two models is that the classical filtering model outputs only one bit, while the multi-output filtering model outputs several bits. We should mention that the technique we used in the multi-output filtering model lies in a more general notion called *subset cryptanalysis*. Recently, Dinur et al successfully applied another type of subset cryptanalysis on discovering a 5-round collision of Keccak. As one can see in Chapters 6 and 7, the advantage of the multi-output model is that it bridges a cryptographic primitive and a set of sequences and Boolean functions. This enables us to make use of the fruitful research outcome in the theory of sequences and Boolean functions. We have demonstrated that the poor performance of a cryptographic primitive under this model will lead to a distinguishing attack of it under the IND-CPA model.

To better measure the performance of TUAK algorithms under this new attack, we also apply it on the ciphers AES, KASUMI and PRESENT. Our study shows that the performance of TUAK's f_1 is similar to AES, both of which have excellent randomness property. It is quite interesting to see that KASUMI and PRESENT is vulnerable to this attack as we could mount this attack on them with a non-negligible success rate. We split the results regarding to the multi-output model into two Chapters. Chapter 6 deals with discovering the cryptographic properties of the cryptographic primitive from its component sequences, and Chapter 7 is from its component Boolean functions.

1.3 Organization of the report

The content of this progress report is organized as follows:

- Chapter 2 provides a description of TUAK's authentication and key generation functions for easy reference;

- Chapters 3 and 4 contain a security analysis of the TUAK algorithm set by considering several known cryptanalytic attacks on message authentication code and key generation functions;

- Chapter 5 presents some some security proofs on the soundness of the construction of the TUAK algorithm set;

- Chapter 6 analyzes the f_1 function in a multi-output filtering model. A distinguisher based on the multi-output filtering model is built for distinguishing a message authentication code of f_1.

- Chapter 7 provides an analysis of the component functions of f_1 in the multi-output filtering model.

- Chapter 8 provides a conclusion and a summary of the work.

Chapter 2

TUAK Algorithm Set for 3GPP

In this chapter we describe all the algorithms of the TUAK algorithm set, which contains eight algorithms for message authentication codes and key derivations. The f_1 and f_1^* are used as message authentication codes, f_2 is used for generating response and f_2, f_4, f_5 and f_5^* are used as key derivation functions.

2.1 Description of TUAK Specification

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 notation which will be used throughout this report. They are the same as those in the specification of TUAK.

- The permutation Keccak-$f[1600]$ is denoted by Π. Throughout this document 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 IN[0],...,IN[1599] and outputs an element of 1600 bits that is represented by OUT[0],...,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 for an easy reference of the security assessment. In this document, we interchangeably use the terms "algorithm" and "function" for the functions in the TUAK algorithm set.

2.1.1 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 that is 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

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

- $\mathrm{IN}[0]\ldots\mathrm{IN}[255] = \mathrm{TOP}[255]\ldots\mathrm{TOP}[0]$;
- $\mathrm{IN}[256]\ldots\mathrm{IN}[263] = \mathrm{INSTANCE}[7]\ldots\mathrm{INSTANCE}[0]$;
- $\mathrm{IN}[264]\ldots\mathrm{IN}[319] = \mathrm{ALGONAME}[55]\ldots\mathrm{ALGONAME}[0]$;
- $\mathrm{IN}[i] = 0$, for $320 \le i \le 511$;
- $\mathrm{IN}[512]\ldots\mathrm{IN}[767] = K[255]\ldots K[0]$ if the length of K is 256 bits;

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

Figure 2.1 depicts an overview of an input assignment to the TOP_C function. The TOP_C function is given by

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

with

$$\mathrm{TOP}_c[0],\ldots,\mathrm{TOP}_c[255] = \mathrm{OUT}[255],\ldots,\mathrm{OUT}[0].$$

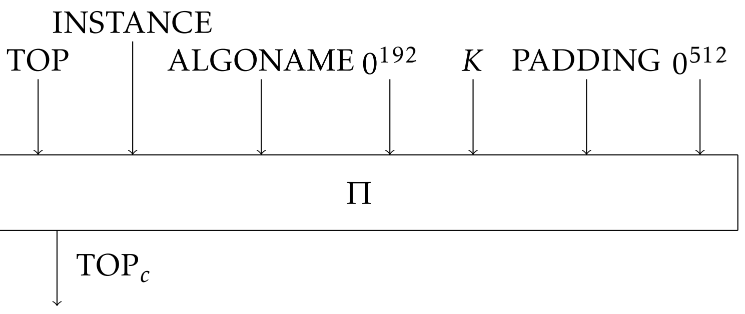

Figure 2.1: The TOP_c function of TUAK

2.1.2 Description of f_1

The function f_1 is used to generate the message authentication codes (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 subsriber key K (128 or 256 bits). The output value MAC can be 64, 128 and 256 bits. Precisely:

- The value of INSTANCE is:

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

- $\text{IN}[0] \ldots \text{IN}[255] = \text{TOP}_c[255] \ldots \text{TOP}_c[0]$;
- $\text{IN}[256] \ldots \text{IN}[263] = \text{INSTANCE}[7] \ldots \text{INSTANCE}[0]$;
- $\text{IN}[264] \ldots \text{IN}[319] = \text{ALGONAME}[55] \ldots \text{ALGONAME}[0]$;
- $\text{IN}[320] \ldots \text{IN}[447] = \text{RAND}[127] \ldots \text{RAND}[0]$;
- $\text{IN}[448] \ldots \text{IN}[463] = \text{AMF}[15] \ldots \text{AMF}[0]$;
- $\text{IN}[464] \ldots \text{IN}[511] = \text{SQN}[47] \ldots \text{SQN}[0]$;
- $\text{IN}[512] \ldots \text{IN}[767] = K[255] \ldots K[0]$ if the length of K is 256 bits ;
- $\text{IN}[512] \ldots \text{IN}[639] = K[127] \ldots K[0]$ if the length of K is 128 bits ;
- $\text{IN}[i] = 0$ for $640 \le i \le 767$ if the length of K is 128 bits ;
- $\text{IN}[i] = 1$ for $768 \le i \le 772$;
- $\text{IN}[i] = 0$ for $773 \le i \le 1086$;
- $\text{IN}[1087] = 1$;

- $\text{IN}[i] = 0$ for $1088 \leq i \leq 1599$.

A high-level overview of the input assignment is provided in Figure 2.3. The MAC function f_1 is defined as

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

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

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

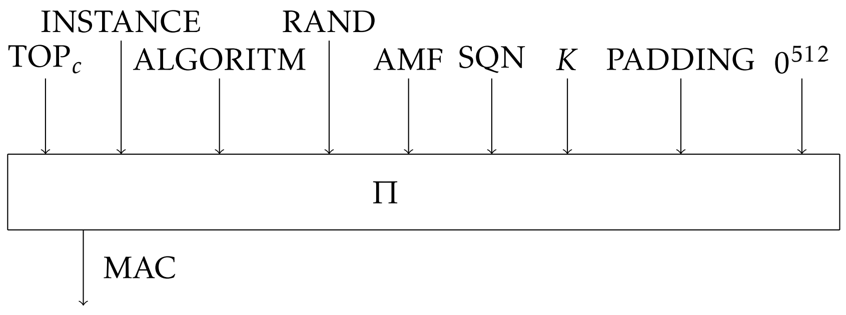

Figure 2.2: The f_1 function of TUAK for generating MAC

2.1.3 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.

- The value of INSTANCE is:

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

- $\text{IN}[0] \ldots \text{IN}[255] = \text{TOP}_c[255] \ldots \text{TOP}_c[0]$;
- $\text{IN}[256] \ldots \text{IN}[263] = \text{INSTANCE}[7] \ldots \text{INSTANCE}[0]$;
- $\text{IN}[264] \ldots \text{IN}[319] = \text{ALGONAME}[55] \ldots \text{ALGONAME}[0]$;
- $\text{IN}[320] \ldots \text{IN}[447] = \text{RAND}[127] \ldots \text{RAND}[0]$;
- $\text{IN}[i] = 0, 448 \leq i \leq 511$;
- $\text{IN}[512] \ldots \text{IN}[767] = K[255] \ldots K[0]$ if the length of K is 256 bits ;
- $\text{IN}[512] \ldots \text{IN}[639] = K[127] \ldots K[0]$ if the length of K is 128 bits ;
- $\text{IN}[i] = 0$ for $640 \leq i \leq 767$ if the length of K is 128 bits ;
- $\text{IN}[i] = 1$ for $768 \leq i \leq 772$;
- $\text{IN}[i] = 0$ for $773 \leq i \leq 1086$;
- $\text{IN}[1087] = 1$;
- $\text{IN}[i] = 0$ 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

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

The output of $f_2 = \text{RES}$, where:

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

The output of $f_3 = \text{CK}$, where:

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

The output of $f_4 = \text{IK}$, where:

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

The output of $f_5 = \text{AK}$, where:

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

A high level overview of the functions f_2, f_3, f_4, f_5 is given below.

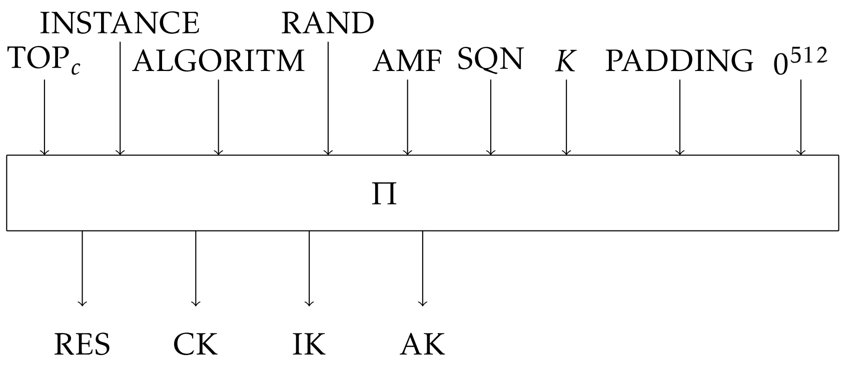

Figure 2.3: The f_i function of TUAK for generating RES, CK, IK, AK

2.1.4 Description of f_5^*

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

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

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

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

where

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

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

2.2 Viewed as a Multi-Output Function

We denote the TUAK algorithm set for authentication and key derivation functions by

$$\mathcal{TUAK} = \{TOP_C, f_1, f_1^*, f_2, f_3, f_4, f_5, f_5^*\}.$$

In the above, we have provided the definitions of TOP_C, f_1, f_1^*, $f_i, 2 \leq i \leq 5$, and f_5^*. In this report, we use $\mathcal{TUAK}$ to denote the TUAK algorithm set. An algorithm in $\mathcal{TUAK}$ takes an input of 1600 bits and outputs multiple bits. This can be regarded as a *multi-output Boolean function*. Mathematically,

$$f_i : \{0,1\}^{1600} \to \{0,1\}^M \text{ where } M = 32, 64, 128, \text{ or } 256.$$
$$(g_0(x), g_1(x), \ldots, g_{M-1}(x)) = f(x), x \in \mathbb{F}_2^{1600}, f \in \mathcal{TUAK}, \tag{2.1}$$

each g_i is a Boolean function from $\mathbb{F}_2^{1600}$ to $\mathbb{F}_2$. We call each $g_i, 0 \leq i \leq M$, a *component function*.

Let $f \in \mathcal{TUAK}$. Assume that t with $0 \leq t \leq 1600$ is the number of positions in the input of f that are set to constant values. Then an algorithm f can then be regarded as a multi-output function from $\mathbb{F}_2^{1600-t}$ to $\mathbb{F}_2^M$ and each g_i is a function of $(1600 - t)$ variables. For example, when TOP$_c$, INSTANCE, ALGONAME, PADDING are constant and the last 512 bits are zeros, then the f_1 function can be considered as a multi-output function from $\mathbb{F}_2^{1600-1152}$ to $\mathbb{F}_2^M$ and each component function g_i is a Boolean function in 448 variables.

Chapter 3

Security Analysis of TUAK Algorithms: I

In this chapter, we analyze the security of the TUAK algorithm set. Here we mainly focus on differential-style cryptanalytic attacks on message authentication codes and key generation functions. Some generic attacks such as birthday attacks that can be applied to hash functions, message authentication codes and key derivation functions are also taken into consideration. The designers of TUAK have chosen the key lengths of the TUAK algorithms so that the time complexity of an exhaustive key search is impractical.

3.1 Differential and Linear cryptanalysis

Differential cryptanalysis introduced in [8] and linear cryptanalysis introduced in [39] are powerful cryptanalytic tools that can also be applied to hash functions for analyzing their security. They are particularly useful when an attacker wants to find a collision or a near-collision on a hash function. The attack mainly exploits the nonlinear properties of the Sboxes used in the primitive.

3.1.1 Differential cryptanalysis

The design of the functions in TUAK is closely based on the Keccak permutation. It is not difficult to observe that a differential or linear attack on a function in TUAK would immediately lead to the respective attack on Keccak. Recently, a few attacks based on differential cryptanalysis have been developed on reduced-round SHA-3 finalist Keccak. For the convenience of the reader, we provide the complexity of the known differential attacks on Keccak in Table 3.1 from [20]. In Table 3.1, m (C) represents m is the number of rounds of Keccak and C is the time complexity of the attack.

Table 3.1: Complexity of the known collision attacks on round-reduced Keccak: the number of rounds attacked with the corresponding time complexity in parentheses

Keccak-224	Keccak-256	Keccak-384	Keccak-512	Reference
2 (practical)	2 (practical)	-	-	[29, 43]
4 (practical)	4 (practical)	-	-	[23]
-	5 (2^{115})	3 (practical), and 4 (2^{147})	3 (practical)	[20]

It can be seen from Table 3.1 that the complexity of the differential attack on 5-round Keccak is already impractical. Therefore, as what is stated in [5] on the possibility of the differential attack on Keccak, we believe that the functions in TUAK are not vulnerable to the differential attack. To further convince the users of TUAK, we provide a lower bound of the complexity for the full-round differential attack on TUAK's functions. Our analysis uses the results in [18] on the lower bound of the weights of differential trail in Keccak and the fact that a good approximation for the complexity of the differential trail is given by $DP(Q) \approx 2^{wr(Q)}$ (when the weight value is below the width of the permutation), where Q is the differential trail and $wr(Q)$ is its weight. Table 3.2 presents the lower bound of a differential trail of round-reduced Keccak and the same lower bounds also hold for a TUAK function. In Table 3.2, we use LB to abbreviate the lower bound, Q to denote a differential trail, and $wr(Q)$ to denote its weight. Note that the complexities provided in Table 3.2 hold for all functions in

TUAK.

Table 3.2: Lower bound on the complexity of differential attack on functions in TUAK

Round	LB on $wr(Q)$	Complexity	Reference
3	32	2^{32}	[18]
6	74	2^{74}	[18]
24	296	2^{296}	[18]

One may be curious about the difference between the complexities in Table 3.1 and in Table 3.2. Table 3.1 presents the actual complexity of the attack. On the other hand, Table 3.2 presents a lower bound of the complexity of the attack. For instance, the complexity of the differential attack on 5-round Keccak-256 is 2^{115}, while the lower bound of the complexity of the differential attack on 6-round Keccak is only 2^{74}. For any function of TUAK, the actual complexity of the differential attack should be much larger than the bounds presented in Table 3.2.

When we compare the complexity of the differential attack on TUAK with the complexity of the birthday attack in Table 3.6, we can observe that the differential attack is not better than the birthday attack on TUAK.

3.1.2 Linear cryptanalysis

We now analyze the TUAK algorithm set against the linear attack. A linear attack on $f \in \mathcal{TUAK}$ would also lead to an attack on Keccak and vice versa as the TUAK algorithm set is constructed using the Keccak permutation. The authors of [5] estimated the weight of the linear trails of up to 6-round Keccak ([5][Table 3.3]). Again, a good estimation of the complexity of the linear attack is $2^{w_c(Q)}$, where $w_c(Q)$ is the correlation weight (defined in [5][page 24]) of the linear trail Q. According to Table 3.3 of [5], the minimum weight of the linear trail is 46 for 4-round Keccak. A rough estimation on the lower bound of the weight of a linear trail on 24-round Keccak is then 276. Since the existence of a linear trail in Keccak is also implied an existence of a linear trail in TUAK's functions, the lower bound of the complexity of the linear attack on functions in TUAK is 2^{276}. Therefore, the linear attack against TUAK is not a practical attack.

3.1.3 Variants of differential attack

There are many variants of the differential attack, including truncated differential attack, impossible differential attack, differential-linear attack, rectangle attack, integral attack, etc. Considering the design of TUAK's algorithms is based on the Keccak permutation by choosing certain bits to be RAND, SQN, AMF and Key bits, it is not difficult to observe that, if one of the above mentioned attacks is mounted on Keccak, it would be directly an attack on TUAK algorithms. We refer the readers to [5][Section 2.4.7, Section 2.4.9] for detailed claim of the resistance of Keccak to these attacks. As a result, we believe that TUAK is not vulnerable to these attacks.

3.2 Zero-sum distinguisher of TUAK

Zero-sum distinguisher was proposed by Aumasson and Meier in the rump session at Crypto 2009. In [12], Aumasson and Meier first proposed a zero-sum distinguisher against Keccak and their zero-sum distinguisher is built upon the result in Theorem 1. Lai and Duan in [34] improved the complexity of Aumasson and Meier's zero-sum distinguisher.

3.2.1 Known results on zero-sum distinguisher

All of the existing zero-sum distinguishers of Keccak are based on the following results.

Theorem 1 ([12, 34]). *Let f be a function from $\mathbb{F}_{2^n}$ to itself corresponding to the concatenation of m smaller balanced S-boxes, $S_1, \ldots, S_m$, defined over $\mathbb{F}_{2^{n_0}}$, where n_0 and n are two positive integers and $n_0 \leq n$. Let δ_k be the maximal degree of the product of any k coordinates from any one of these smaller S-boxes.*

Then, for any function G from $\mathbb{F}_{2^n}$ into $\mathbb{F}_{2^\ell}$, we have

$$\deg(G \circ F) \leq n - \frac{n - \deg(G)}{\gamma},$$

where $\gamma = \max_{1 \leq i \leq n_0 - 1} \frac{n_0 - i}{n_0 - \delta_i}$.

Applying Theorem 1 on the Keccak permutation, we have the following upper bound of the algebraic degree of $R^i, 1 \leq i \leq 24$, where R is the round function of the Keccak permutation.

Corollary 1. *Let $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$ be the round function of Keccak-f, where $\iota, \chi, \pi, \rho, \theta$ are the component functions defined in [5]. Then for any function G from $\mathbb{F}_{2^{1600}}$ to itself, we have*

$$\begin{aligned} \deg(G \circ R) &= \deg(G \circ \chi) \leq 1600 - \frac{1600 - \deg(G)}{3}, \textit{ and} \\ \deg(G \circ R^{-1}) &= \deg(G \circ \chi^{-1}) \leq 1600 - \frac{1600 - \deg(G)}{2}. \end{aligned}$$

Note that as the notations in Theorem 1, we have $n = 1600$ and $n_0 = 5$ for Keccak.

For the functions in $\mathcal{TUAK}$, Corollary 1 can be applied to build a zero-sum distinguisher. Below we present the complexity of TUAK's zero-sum distinguisher. Assume that $f \in \mathcal{TUAK}$, we may regard it as a permutation by restricting the Keccak permutation on some subspace of $\mathbb{F}_2^{1600}$, and we denoted this subspace by $f_i = \text{Keccak-}f|_{\mathbb{F}_2^m} : \mathbb{F}_2^m \to \mathbb{F}_2^m$, where $\sigma|_S$ denotes the restriction of the function σ on a subset S of its domain. For instance, we may regard f_1 as a permutation over $\mathbb{F}_2^{128}$ by keeping, for example 128 key bits positions independent and fixing all other bits of an input of 1600-bit, as they required to be constants according to the specification.

3.2.2 New results on TUAK's zero-sum distinguisher

An existence of a zero-sum distinguisher on a TUAK algorithm is a distinguishing property. We present the complexity of the zero-sum distinguisher for all functions in TUAK in Table 3.3. In the table, a number in bold implies that the complexity is better than the exhaustive search. In Table 3.3 we can observe that:

- for functions f_1 and f_1^*, when the length of key is 256, a zero-sum distinguisher with complexity 2^{430} (vs the maximal complexity 2^{432}) can be built. The algebraic degree of f_1 and its inverse f_1^{-1} after the i-th round is listed in Table 3.4.

- for functions $f_j, 2 \le j \le 5$ and f_5^*, when the length of key is 256, a zero-sum distinguisher can be developed with complexity 2^{383}, which is better than the maximal complexity 2^{384}. The algebraic degree of f_j and its inverse f_j^{-1} after the i-th round can be found in Table 3.5.

Table 3.3: Complexity of the zero-sum distinguisher for the TUAK algorithms

Parameter	TOP_C	f_1	f_1^*	f_2	f_3	f_4	f_5	f_5^*
$\|K\| = 128$	2^{128}	-	-	-	-	-	-	-
$\|K\| = 256$	2^{256}	-	-	-	-	-	-	-
$\|K\| = 128, \|\text{RAND}\| = 128, \|\text{SQN}\| = 48$	-	2^{304}	2^{304}	-	-	-	-	-
$\|K\| = 256, \|\text{RAND}\| = 128, \|\text{SQN}\| = 48$	-	$\mathbf{2^{430}}$	$\mathbf{2^{430}}$	-	-	-	-	-
$\|K\| = 128, \|\text{RAND}\| = 128$	-	-	-	2^{256}	2^{256}	2^{256}	2^{256}	2^{256}
$\|K\| = 256, \|\text{RAND}\| = 128$	-	-	-	$\mathbf{2^{383}}$	$\mathbf{2^{383}}$	$\mathbf{2^{383}}$	$\mathbf{2^{383}}$	$\mathbf{2^{383}}$

The complexities of the zero-sum distinguisher presented in Table 3.3 show that the zero-sum distinguishing attack is impractical against TUAK. Finally, we should mention that the zero-sum distinguishing attack will not affect the security of the TUAK's algorithm set.

3.3 Differential distinguishing attack

Kim et al. [33] proposed a differential distinguishing attack on a HMAC-based message authentication code. The attack exploits the differential property of the underlying hash function. Let q be the probability that one finds a collision in the compression function of the hash function. The

Table 3.4: Upper bounds of the degree of f_1 and f_1^{-1} after i-th round: Case $|K| = 256$

No of round	Upper bound (forward)	Upper bound (backward)
1	2	3
2	4	9
3	8	27
4	16	81
5	32	243
6	64	337
7	128	384
8	256	408
9	373	412
10	412	426
11	425	429
12	429	430
13	431	431

differential distinguishing attack works as follows: a) an adversary collects $2{\cdot}q^{-1}$ randomly chosen messages M_i and then computes $M_i' = M_i \oplus \Delta$ where Δ is associated with the compression function and has the length same as that of M_i; b) Adversary then computes $\text{MAC}_i = MAC_K(M_i)$ and $\text{MAC}_i' = MAC_K(M_i')$; and c) If $\text{MAC}_i \oplus \text{MAC}_i' = 0$, output MAC_i. This is nothing but forging a MAC.

Note that the construction of HMAC in [33] is different from the construction of TUAK. We analyze the TUAK algorithm set against the differential distinguishing attack. In order to successfully launch a differential distinguishing attack on TUAK, one must find a good differential characteristic on the Keccak permutation with the following inputs at their respective positions by: a) keeping same the values of INSTANCE, ALGOTITHM, TOP_c of TUAK; b) the padding PADDING of TUAK; and c) the last 512 bits are zero. Finding a good differential characteristic under these conditions can be regarded as a constrained-input and constrained-output (CICO) problem [6]. According to the CICO problem, it is hard to obtain a good differential characteristic with probability greater than

Table 3.5: Upper bounds of the algebraic degree of f_j and f_j^{-1} ($j \in \{2,3,4,5\}$) after i-th round: Case $|K| = 256$

No of round	Upper bound (forward)	Upper bound (backward)
1	2	3
2	4	9
3	8	27
4	16	81
5	32	232
6	64	307
7	128	345
8	256	364
9	340	373
10	368	378
11	378	380
12	381	381
13	382	382

$2^{-\frac{|M|}{2}}$ where M is the length of the output a function in the TUAK algorithm set. Moreover, in Section 3.1.1, we have seen that the upper bound of the probability of a differential characteristic of TUAK is 2^{-296}, which is much lower than $2^{-\frac{|M|}{2}}$ with $M = 256$. Thus, the complexity of the differential distinguishing attack on TUAK is not better than the complexity of birthday-type attack, which is $2^{\frac{|M|}{2}}$.

3.4 Boomerang attack

The boomerang attack was proposed by D. Wagner against block ciphers by finding and combining short differential paths, instead of using a long differential path [48]. Since the TUAK algorithm set is designed using the Keccak permutation, we investigate the boomerang attack on the TUAK algorithm set by exploiting the differential paths of the Keccak permutation.

3.4.1 A general setting of boomerang attack on TUAK

Assume that Π_1 and Π_2 are two sub-permutations that decompose the Keccak permutation, i.e., $\Pi = \Pi_1 \circ \Pi_2$. Since the round constants of the Keccak permutation are different for different rounds, the sub-permutations Π_1 and Π_2 are different. To apply a boomerang attack on a TUAK algorithm, one needs to find two differential paths of the Keccak permutation Π – one differential path $\mathsf{DP}_s : \Delta \rightarrow \Delta'$ with probability p_s is for Π_1 and the second differential path $\mathsf{DP}_e : \nabla' \rightarrow \nabla$ with probability is for Π_2 – instead of finding a long differential path. The algorithms of the TUAK algorithm set accept an input of a specific format. As a result, the input difference for a differential characteristic has be to chosen accordingly. We showed the types of Δ and ∇ below. Since TUAK uses the Keccak permutation to build its algorithms, a boomerang attack can be described as that of block ciphers. We describe a boomerang attack on f_1 with output 256 bits as follows. Recall that the f_1 algorithm accepts an input of the form. $P_1 = \text{TOP}_c \| \text{INSTANCE} \| \text{ALGONAME} \| \text{RAND} \| \text{AMF} \| \text{SQN} \| \text{K} \| \text{PADDING} \| 0^{512}$.

$$\begin{aligned}
P_2 &= P_1 \oplus \Delta, && \Delta = 0^{320} \| \{0,1\}^{192} \| \{0,1\}^{128} \| 0^{960} \text{ for 128 bits key} \\
P_2 &= P_1 \oplus \Delta, && \Delta = 0^{320} \| \{0,1\}^{192} \| \{0,1\}^{256} \| 0^{832} \text{ for 256 bits key} \\
C_1 &= \Pi(P_1), && C_2 = \Pi(P_2) = \Pi(P_1 \oplus \Delta) \\
C_3 &= C_1 \oplus \nabla, && C_4 = C_2 \oplus \nabla \\
P_3 &= \Pi^{-1}(C_3), && P_4 = \Pi^{-1}(C_4)
\end{aligned}$$

and $P_3 \oplus P_4 = \Delta$ hold with probability $p_s^2 p_e^2$. In the above, ∇ is chosen in such a way that the first 256 bits of C_3 and C_4 are the same and Δ is chosen so that the input messages P_1 and P_2 satisfy the input message format. All the above conditions for the boomerang attack are the same as that of a block cipher except the condition on ∇ that the first 256 bits of C_3 and C_4 are the same. The input assignment in the Keccak permutation for obtaining TUAK algorithms makes difficult to find differential characteristics DP_s DP_e of TUAK for Δ and ∇', respectively. The diffusion property of the inverse of the diffusion layer of the Keccak permutation is better than that of the diffusion layer. Moreover, the algebraic degree of the inverse of Keccak's Sbox is 3 and the differential property of the inverse Sbox is better [7]. As a result, it is hard to find good differential characteristics for Π_2 as well as its inverse. According to [18], a upper bound on the probability of the differential characteristic for the first half of the Keccak permutation is 2^{-184}. Thus, the lower bound of the complexity of a successful boomerang attack on f_1 is $O(2^{4 \cdot 148}) = O(2^{592})$, which is not better than a birthday attack. Thus the f_1 function of TUAK is resistant to the boomerang attack.

Joux and Peyrin in [31] adapted the boomerang attack on block ciphers to iterated hash functions as an improvement of the collision attack on SHA-1. The starting point of the attack is that it requires a differential characteristic for the iterated hash function and an auxiliary differential path is used to improve the complexity of the attack. We believe that it would be hard to find a good auxiliary differential characteristic with probability greater than 2^{-296} for the TUAK algorithm set. Hence, the TUAK algorithm set will resist the boomerang attack.

3.5 Birthday attack

A birthday attack is a generic attack that can be applied to any cryptosystem and which has been discussed in [42]. In this section we provide the complexity of the birthday attack on the TUAK algorithm set. For the TOP_C function, the output is only dependent on the key and TOP as other inputs to the TOP_C function are constant. The TOP_C function accepts the keys of length 128 bits and 256 bits and outputs TOP_c of length 256 bits. For a fixed key, the TOP_c is constant. If one wishes to find the same TOP_c value for two different keys, then by the birthday paradox, one needs to query a random oracle for about $O(2^{128})$ keys with $|K| = 128$ and 256, as TOP is fixed by the operator. When the length of the key of TOP_C is 128 bit, the complexity of the birthday attack is not better than the complexity of the exhaustive key search.

For a user, the key of f_1 is fixed. Using f_1 function, one can obtain a message authenticate code (MAC) for a RAND, SQN, AMF. To forge a MAC of length M, an attacker can make 2-tuple $(RAND, SQN)$ queries to a random oracle because AMF is fixed in the protocol. By the birthday paradox, the attacker needs to make $O(2^{\frac{M}{2}})$ $(RAND, SQN)$-tuple queries and obtain their MACs to forge a MAC. Thus the time complexity of the birthday attack is $O(2^{\frac{M}{2}})$ and the data complexity is $O(M2^{\frac{M}{2}})$. In particular, the time complexity of forging a MAC of 64-bit produced by f_1 is $O(2^{32})$. Similarly, the time complexities for forging a MAC of 128 bits and 256 bits are $O(2^{64})$ and $O(2^{128})$, respectively.

Similar to the above, the birthday attack can also be applied to the response function f_2 and the key derivation functions f_3, f_4, f_5 and f_5^*. In f_2 to f_5^* functions, only the random number varies, the key is fixed and other inputs are constants for a particular instance. For f_2 function, an attacker can produce the same response RES for two different random numbers by applying the birthday attack with complexity $O(2^{\frac{M}{2}})$ where M is the length of a response. We present near-collisions and collisions on f_2 with output 32 bits below. Applying the birthday attack on f_3 (or f_4), one can produce the same cipher key (or integrity key) of length M for two different random numbers with complexity $O(2^{\frac{M}{2}})$. If two different random numbers produce the same cipher key (or integrity key), an attacker only needs to observe the random numbers in different protocol sessions. If one of two random numbers matches, then the attacker can easily gets the cipher key (or integrity key) without knowing the secret key of f_3 (or f_4).

By applying a birthday-style attack on f_5 (or f_5^*), the same anonymity key can be produced for two different random numbers with complexity $O(2^{24})$. Note that a successful forgery of an anonymity key leads to a successful recovery of SQN in the protocol. We present a near-collision with Hamming weight 5 on f_5 below. A summary of the time complexity of the birthday attack on the TUAK algorithm set is provided in Table 3.6.

Table 3.6: Complexity of the birthday attacks on TUAK algorithms

Algorithm	TOP_C	f_1-64	f_1-128	f_1-256	f_2-32	f_2-64	f_2-128
Complexity	$O(2^{128})$	$O(2^{32})$	$O(2^{64})$	$O(2^{128})$	$O(2^{16})$	$O(2^{32})$	$O(2^{64})$

Algorithm	f_2-256	f_3-128	f_3-256	f_4-128	f_4-256	f_5-48	f_5^*-48
Complexity	$O(2^{128})$	$O(2^{64})$	$O(2^{128})$	$O(2^{64})$	$O(2^{128})$	$O(2^{24})$	$O(2^{24})$

3.5.1 Practical near-collision on f_2-32

Here we present near-collisions on f_2 with output 32 bits. We implemented a birthday-type attack to find near-collisions of f_2. First, we randomly generate a key of length 128 bits to the f_2 function. Then, we generate a population with size P of random numbers and compute the responses using f_2 for each random number. If two responses for two different random numbers mismatch at one position, we call it a near-collision with Hamming weight one for f_2. For $P \leq 2^{14}$, we have found a number of near-collisions for f_2 and we present one such collision below. Note that the complexity of the near-collision search is lower than the complexity of the birthday attack.

$$\begin{aligned}
K &= \mathtt{0x679C26F9D43CCC83DB093ED88734EE49} \text{ (128 bits)}\\
RAND_1 &= \mathtt{0x8FD0FE77C7D38C6008BEFF9B3572A110} \text{ (128 bits)}\\
RAND_2 &= \mathtt{0xBBB0F4C8E303A53A948E0BCC9A896E1B} \text{ (128 bits)}\\
RES_1 &= \mathtt{0xEB802BC1} \text{ (32 bits)}\\
RES_2 &= \mathtt{0xEB802BC0} \text{ (32 bits)}
\end{aligned}$$

3.5.2 Practical collision on f_2-32

We again apply the birthday attack to find collisions on f_2-32. The attack works in the similar way as described above. If f_2 generates the same response (RES) for two different random numbers, we call it a collision on f_2-32. Similar to the above, for $P \leq 2^{14}$, we also have found many collisions on f_2-32 and present a collision below. The reason behind successfully finding a collision or near-collision is due to the short length of RES.

$$\begin{aligned}
K &= \mathtt{0x679C26F9D43CCC83DB093ED88734EE49} \text{ (128 bits)}\\
RAND_1 &= \mathtt{0x2F2C747DB960C98FCAC8868ACD34087A} \text{ (128 bits)}\\
RAND_2 &= \mathtt{0x0EEEC50DA872D1E7AF0598D5C6FC32E3} \text{ (128 bits)}\\
RES_1 = RES_2 &= \mathtt{0xB060C7B4} \text{ (32 bits)}
\end{aligned}$$

Table 3.7: Summary of collision attacks on f_2 (according to the birthday attack)

	f_2-32	f_2-64	f_2-128	f_2-256
Complexity	$2^{14} (< 2^{16})$	2^{32}	2^{64}	2^{128}
Decision	Insecure	Secure	Secure	Secure
Comments	Collision found	–	–	–

3.5.3 Near-Collision on f_5

Although the random number RAND in the protocol in Figure 1.1 is known, we want to test the collision resistance property of f_5 function for different random numbers. We perform a birthday-type attack on f_5 for finding collisions or near-collisions. We kept the key of f_5 fixed and search for different two different random numbers so that f_5 produces near-collisions or collisions. Other inputs of f_5 take the specified values provided in the TUAK algorithm set. We tested for around 2^{21} random numbers and found many near-collisions of f_5. A near-collision with Hamming weight 3 is presented below. Note that a successful recovery of anonymity key (AK) leads to the recovery of sequence number (SQN) in the protocol. Again, a successful finding of near-collisions is due to the short length of the anonymity key. The Hamming weight of $AK_1 \oplus AK_2$ is 3.

$$
\begin{aligned}
K &= \text{0x679C26F9D43CCC83DB093ED88734EE49 (128 bits)}\\
RAND_1 &= \text{0xBDF38957F4F826547EA688A4E147E96E (128 bits)}\\
RAND_2 &= \text{0x6D177520ACAAF56FBE9A995CA9CCA98E (128 bits)}\\
AK_1 &= \text{0xD75C90344CD0 (48 bits)}\\
AK_2 &= \text{0xD75C90344CC9 (48 bits)}
\end{aligned}
$$

3.6 Key search complexity reduction with structural property of TUAK

From the definition of $f_2 \sim f_5$, we can observe that the input of $f_2 \sim f_5$ is the same for a fixed length of RES, CK, IK and K and the outputs of $f_2 \sim f_5$ are taken from different output positions of Π. Assume that an adversary knows RES, CK, IK for a random number number RAND and she does not known the key. If the adversary wishes to recover the key K from the known CK, then she can search for the same CK by fixing RAND and varying the key K in f_3 function. While varying different keys in f_3, the adversary can get many keys for which the CK is the same due to the collision, but it is hard for the adversary to decide the correct key. When the RES and IK is known to the adversary, she can easily decide the correct key by computing RES and IK for a key and RAND and these with the known RES and IK. This may reduce the complexity of searching the correct key. However, the key search complexity in worst case is $2^{|K|}$ where $|K| = 128$ or 256. If the INSTANCE values of $f_2 \sim f_5$ are different for a fixed length of RES, CK, IK and K, then, by the above technique, the adversary would not have advantage of deciding the correct key with complexity less than $O(2^{|K|})$.

3.7 Summary

In this chapter, we mainly focused on attacks based on differential cryptanalysis, which exploit the differential paths of Keccak's permutation. We also studied the birthday attack on TUAK algorithms. Some practical near-collisions and collisions on f_2-32 and f_5 algorithms are presented. We believe that the (near) collisions occurred due to the short output length of f_2-32 and f_5 algorithms.

Chapter 4

Security Analysis of TUAK Algorithms: II

In this chapter, we consider interpolation attack, algebraic attack, cube attack, differential-style key-recovery attack, slide attack, and extension attack on the TUAK algorithms.

4.1 Interpolation attack

Interpolation attack on block ciphers was proposed by Jakobsen and Knudsen in [30]. The complexity of an interpolation attack depends on the degree of the polynomial approximation and the number of nonzero terms in the polynomial approximation [30, 51]. Since the algebraic degree of Keccak's S-box is only 2, it is a natural question whether there is any influence on TUAK' functions by considering techniques in the interpolation attack.

First, as demonstrated in Tables 3.4 and 3.5, all the functions in the TUAK algorithm set have the maximal algebraic degree. Recall that Keccak uses the Sbox χ over $\mathbb{F}_{2^5}$. Second, we computed all the polynomial representations of χ for different defining polynomials of degree 5 of $\mathbb{F}_{2^5}$ and counted the number of nonzero coefficients in the polynomial representations of χ. A polynomial over $\mathbb{F}_{2^5}$ can have at most 32 terms. Our computational results show that the number of terms in a polynomial representation of χ takes one of the values from the set $\{23, 24, 25\}$ and the Sbox χ is not affine equivalent to a function with few nonzero terms. As a result, the TUAK algorithm set is not vulnerable to the interpolation attack.

4.2 Experiment on component functions

In this section, we provide some analysis on cryptographic properties of component functions of the TUAK algorithms for small parameters. Especially, we compute the algebraic degree, algebraic immunity, and the number of terms in the algebraic representation of a component function. We conduct two experiments: the first experiment is about calculating the exact algebraic degree and algebraic immunity, and the second experiment is about counting the number of variables appear in a component function of the Keccak permutation.

4.2.1 Experiment A: Cryptographic properties of component functions

Here we first explain how the component functions in n variables are constructed using TUAK. The input assignment of TUAK for constructing the component functions in our experiment is as follows. We use the same input assignment of TUAK for TOP_c, ALGORITHM, PADDING and the last 512 bits are all zero specified in the specification. We choose the first n positions from key bit positions which accept all possible 2^n inputs, and other key bit positions are set to zero. The RAND, AMF and SQN positions are set to all one. An overview of the input assignment is provided in Figure 4.1. For this choice of input assignment to TUAK, we can construct a set of component functions using Eq. (2.1) and we denote the set of component functions by $\{g_0, ..., g_{255}\}$

We compute the following cryptographic properties of component functions $g_0, g_1, ..., g_{255}$ in n variables:

1. the algebraic degree of the ANF representation of g_i

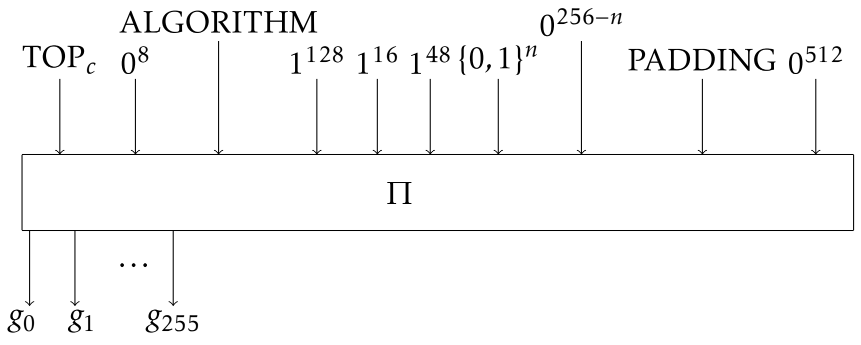

Figure 4.1: Component functions of TUAK in n variables

2. the algebraic immunity of g_i
3. the nonlinearity of g_i
4. the number of nonzero terms of in the ANF of $g_0, g_1, ..., g_{255}$.

We have computed the above properties for the component functions when $n = 8, 9, ...,$ and 13 and presented the results in Tables 4.1 and 4.2. The nonlinearity of g_i in 8-variable is provided in Table 4.2, where "NL" denotes the nonlinearity and "No." denotes the index of component function g_i. The number of nonzero terms in the ANF representation of g_i's is different. Thus, we provide upper and lower bounds of the numbers of terms in Table 4.1. For the nonlinearities of the component functions of TUAK for $n = 9, ..., 13$, see our technical report. We also study the cryptographic properties of the component functions g_i's which are obtained by setting RAND, AMF, SQN to zero and other input assignments are the same as above. The cryptographic properties of the component functions are similar to the results in Tables 4.1 and 4.2.

4.2.2 Experiment B: Number of variables in component functions

Recall that the Keccak permutation is defined from $\mathbb{F}_2^{1600}$ to $\mathbb{F}_2^{1600}$ and is constructed by iterating its round function 24 times. Each component

Table 4.1: Cryptographic properties of component functions of TUAK

n	Algebraic degree	Algebraic immunity	# of nonzero terms
8	7/8	4	$106 \sim 153$
9	8/9	5	$233 \sim 287$
10	9/10	5	$468 \sim 557$
11	10/11	6	$950 \sim 1091$
12	11/12	6	$1968 \sim 2158$
13	12/13	7	$3957 \sim 4213$

function of the Keccak permutation can be considered as a function from $\mathbb{F}_2^{1600}$ to $\mathbb{F}_2$. We perform an experiment to count the presence of the number of variables, out of 1600, in a component function over different rounds while constructing the Keccak permutation. Our experiment shows that, after four iterations of Keccak's round function, all 1600 variables appear in each component function due to the good diffusion property of Keccak. Therefore, a component function of the Keccak permutation would contain all 1600 variables. Consequently, all independent variables in the input of TUAK will appear in each component function of TUAK. This is also validated by **Experiment A**.

4.3 Algebraic attack

Algebraic attack [16] is a powerful cryptanalytic attack for recovering the key of a cryptographic primitive. The algebraic attack consists of two phases. First, a system of multivariate equations in key bits is constructed. Second, the system of equations is solved for recovering the key bits by linearization, eXtended spare linearisation, Gröbner basis algorithms, or SAT-solvers.

The algebraic attack can be applied to the TUAK algorithms TOP_C and $f_1 \sim f_5^*$. We note that all the TUAK algorithm set use the same secret key for a fixed used for generating TOP_c, MAC, RES, CK, IK, and AK. In the algebraic attack, one can consider each key bit is an element of $\mathbb{F}_2$ and construct a system of equations over $\mathbb{F}_2$. Assume that the length of the

Table 4.2: Nonlinearity of component functions in 8 variables

No.	NL	No.	NL	No.	NL	No.	NL	No.	NL	No.	NL	No.	NL	No.	NL	No.	NL	No.	NL
1	106	2	107	3	100	4	105	5	100	6	105	7	102	8	103	9	95	10	106
11	98	12	102	13	106	14	105	15	104	16	103	17	102	18	104	19	106	20	103
21	94	22	103	23	98	24	108	25	108	26	104	27	104	28	104	29	104	30	104
31	105	32	103	33	100	34	106	35	108	36	104	37	106	38	103	39	99	40	102
41	102	42	107	43	107	44	103	45	102	46	107	47	103	48	104	49	104	50	104
51	100	52	99	53	105	54	103	55	99	56	103	57	107	58	105	59	105	60	104
61	106	62	103	63	100	64	104	65	102	66	101	67	104	68	103	69	100	70	103
71	100	72	104	73	109	74	103	75	104	76	100	77	104	78	108	79	93	80	102
81	100	82	104	83	99	84	101	85	102	86	104	87	104	88	105	89	104	90	102
91	102	92	108	93	101	94	101	95	102	96	105	97	104	98	101	99	104	100	101
101	106	102	106	103	102	104	106	105	105	106	103	107	101	108	102	109	103	110	103
111	98	112	104	113	103	114	105	115	104	116	108	117	99	118	105	119	103	120	105
121	94	122	99	123	101	124	104	125	106	126	108	127	99	128	104	129	105	130	106
131	105	132	106	133	102	134	106	135	102	136	103	137	106	138	107	139	103	140	103
141	106	142	108	143	104	144	103	145	107	146	101	147	103	148	102	149	106	150	106
151	106	152	105	153	103	154	103	155	100	156	109	157	104	158	106	159	105	160	109
161	107	162	103	163	105	164	104	165	106	166	102	167	106	168	103	169	105	170	106
171	103	172	108	173	103	174	102	175	104	176	104	177	106	178	105	179	103	180	106
181	102	182	104	183	104	184	101	185	102	186	109	187	104	188	107	189	107	190	109
191	101	192	98	193	106	194	104	195	103	196	105	197	97	198	105	199	101	200	103
201	105	202	104	203	101	204	101	205	106	206	105	207	109	208	105	209	101	210	104
211	97	212	108	213	105	214	103	215	106	216	104	217	101	218	105	219	103	220	105
221	107	222	102	223	101	224	105	225	108	226	107	227	104	228	105	229	102	230	105
231	102	232	104	233	105	234	104	235	107	236	102	237	100	238	99	239	107	240	102
241	103	242	107	243	102	244	103	245	96	246	105	247	105	248	106	249	99	250	100
251	107	252	103	253	106	254	103	255	107	256	105								

key is $|K|$. For the TUAK algorithm set, $|K| = 128$, or 256. Any function of the TUAK algorithm set can then be regarded as a multi-output Boolean function in $|K|$ variables, as other input positions are fixed. A system of multivariate equations in key bits can be formed in different ways, such as one can consider a single algorithm, multiple algorithms or all algorithms in the TUAK algorithm set since all the algorithms have the same key. An adversary can construct a system of equations as follows:

- using only one function f_i for different random numbers;
- using a set of functions f_i's for different random numbers;
- using all functions f_i's for different random numbers, as all the functions have the same secret key.

If the adversary uses only one function f_i, she needs to construct equations for many random numbers and the number of equations constructed every time depends on the number of outputs of f_i. On the other hand, the number of equations an adversary constructs each time for all functions

f_i's is less than the number of equations an adversary constructs for a f_i function.

In [13], Boura et al. showed that the upper bound of the degree of the Keccak permutation is achievable at the 16-th round. We conducted **Experiment B** to keep track of the number of variables present in a component function of the Keccak permutation. Our experiment shows that, at the 4-th round, all 1600 input variables of Keccak present in all component functions. According to **Experiment A** in Section 4.2, our results show that the algebraic degree and algebraic immunity of a component function of TUAK is maximum for $n = 8, \ldots,$ and 13. Since TUAK is constructed based upon Keccak and according to our experiments for small parameters, it is expected that all component functions of TUAK would have the maximum algebraic degree and high algebraic immunity. This analysis states that it is hard to construct a system of multivariate equations in key bits of TUAK algorithms for $|K| = 128$ and 256.

Even if one is successful in constructing a system of equations, the number of unknowns for the eXtended linearization phase is $T = \sum_{i=0}^{|K|} \binom{|K|}{i} = 2^{|K|}$ where $|K|$ is the length of the secret key. The time complexity to recover the key bits by solving the system of linear equations using the Strassen algorithm is $O(T^w)$ with $w = \log_2 7$ where the attacker needs to collect T equations. Table 4.3 presents the complexities of the algebraic attack for different functions of TUAK. The complexity of the algebraic attack in Table 4.3 shows that it is not better than the exhaustive key search attack. Thus, TUAK is resistant to the algebraic attacks.

Table 4.3: Complexity of the algebraic attacks

Key size	TOP_C	f_1	f_1^*	f_2	f_3	f_4	f_5	f_5^*
128	$O(2^{128 \cdot w})$	$O(2^{128 \cdot w})$	$O(2^{128 \cdot w})$	$O(2^{128 \cdot w})$	$O(2^{128 \cdot w})$	$O(2^{128 \cdot w})$	$O(2^{128 \cdot w})$	$O(2^{128 \cdot w})$
256	$O(2^{256 \cdot w})$	$O(2^{256 \cdot w})$	$O(2^{256 \cdot w})$	$O(2^{256 \cdot w})$	$O(2^{256 \cdot w})$	$O(2^{256 \cdot w})$	$O(2^{256 \cdot w})$	$O(2^{256 \cdot w})$

4.4 Cube attack

Cube attack [19] is a key-recovery attack that can be applied to any cryptosystem, and a cube tester is used to detect the nonrandom behavior of a cryptographic function [2]. In the cube attack, the input of a primitive consists of a set of secret variables and another set of public variables. The cube attack works as follows. First, a system of linear or quadratic equations in the secret variables is constructed by exhausting different cube dimensions in public variables. Then, the system of equations is solved to recover the secret key bits. To apply the cube attacks on TUAK algorithms $f_1 \sim f_5^*$, we only need to decide the bits positions of RAND as the public variables, since the secret key bits are fixed and other bit positions are constant. Note that the cube attack cannot be applied to TOP_C as all of its inputs are fixed and the TOP value is fixed and chosen by the operator. Again, to apply the cube attack on f_1 function with a MAC of size M, one can choose the bit positions of RAND as public variables (initial vector) and one of the component functions (out of M functions) as an output. Similarly, for $f_2 \sim f_5^*$ algorithms, the key bits are the secret variables and the RAND can be considered as public variables and other inputs are fixed, but constants.

Cube attack is successful when the algebraic degree of the internal state transition of a cryptoalgorithm grows slowly, like the state update of the stream cipher Trivium [19]. One would be successful in applying the cube attack to TUAK when one successfully launches the cube attack to Keccak, as the design of TUAK is based upon Keccak's permutation. According to Table 1 in [13], it can be observed that the upper bound of the algebraic degree of the Keccak permutation grows exponentially with the number of rounds till 10 rounds and it achieves the maximum algebraic degree at the 16-th round. According to **Experiment B**, at the 4-th round, all 1600 variables appear in each component function. Thus, it is believed that the algebraic degree of the component functions of TUAK will be maximum and all input variables will appear in each component function. In [36], Lathrop studied the cube attack on 4-round Keccak-224 in a message authenticate code setting and found 112 cubes (maxterms of degree less than or equal to 12) with linearly independent superpolys on secret variables. Recently, Dinur et al. in [24] presented a practical cube attack on 5-round Keccak in a message authenticate code setting and 6-round Keccak in a stream cipher mode for recovering keys of the primitive.

The experiment in Section 4.2 shows that all the component functions have the maximum algebraic degree and the number of monomials in the ANF representation lies in the range of 114 and 155. Since TUAK uses the Keccak permutation and its algebraic degree is maximum and all variables appear in all component functions, it would hard to find linear superpolys on the key bits. For 128-bit or 256-bit key, it would hard to construct a system of linear or quadratic equations in key bits to recover the key.

4.5 Slide attacks

Slide attacks, proposed by Biryukov and Wagner [10], were developed on a block cipher by exploiting weaknesses in the key scheduling and in the general structure of the cipher. Such an example of the weakness is that the key scheduling produces round keys in a cyclic fashion, as a result, the cipher is vulnerable to the known-plaintext attack. To investigate the slide attack resistance property to TUAK, we must study the slide attack resistance property of the Keccak permutation. The designers of Keccak studied the slide attack on Keccak. According to the designers, without the round constant part, the round function of Keccak's permutation is a symmetric function, as a result, the Keccak permutation would have the symmetry property. However, the Keccak permutation has a round constant addition phase and the round constants are different at each round. This destroys the symmetry property of the Keccak permutation.

The output of the TOP_C function is constant when the key K is fixed. For TUAK's key derivation functions, a cipher key, integrity key, or anonymity key is derived by using the key K and a random number. For a fixed key K, the key derivation functions f_3, f_4 and f_5 (or f_5^*) of TUAK produce keys CK, IK, and AK, respectively in a cyclic fashion when the random numbers repeated cyclically or a collision occurred, as TUAK is constructed by the Keccak permutation. Unlike block ciphers, TUAK does not have any key scheduling algorithm. Therefore, the slide attack on TUAK cannot be applied.

Gorski, Lucks and Peyrin proposed a slide attack on sponge-function like hash structures and the attack can also be applied to message authentication codes based on sponge function like hash function structures [28]. The MAC is defined as $MAC(K,M) = H(K\|M)$, where H is a sponged-based hash function, K is the key and M is the message. In the slide attack on such a MAC, an attacker needs to find a slid pair (C_i, C_j) such that

$$C_i = C_i^1 \| C_i^2 \| \cdots \| C_i^l$$
$$C_j = C_i \| C_j^{l+1} \text{ with } X^{k+l+r+1} = F(X^{k+l+r+1}) \text{ and } X^i = F(S(X^{i-1}, C_i))$$

where F is the round function and S defines how the message is incorporated in the internal state [28]. None of the algorithms of TUAK accepts an input of length greater than 1600 bits. For the f_1 algorithm, the strategy of the above slide attack cannot be applied as f_1 does not accept an input of length greater than 1600, as a result, it is hard to find a slid pair.

4.6 Key-recovery attacks

In this section we discuss key recovery attacks on the TUAK algorithm set. A successful key recovery attack on TUAK is able to recover the key of a TUAK algorithm. Contini and Yin in [15] presented a partial key recovery attack on HMAC/NMAC-MD4/MD5/SHA0 using collisions of MD4, MD5, and SHA0. Their key-recovery attack uses a good differential path for the underlying compression function that leads to a real collision and the collision path allows to learn many key bits of HMAC/NMAC-MD4. Later on, Fouque et al. in [25] proposed a key recovery attack on HMAC/NMAC-MD4 that can recover the full key of MAC. The attack is also based on collision search techniques. It finds a good IV-dependent differential path of MD4 for a message difference and computes a set of sufficient conditions. The attack first finds at least a pair of message that produces one collision and then a plenty of messages related to the first one are constructed and that produce collisions. The attacker learns the key by examining message pairs for the collisions.

We apply the above two key-recovery attacks against the TUAK algorithm set. We note that the basic requisite for the above two attacks is a good differential characteristic of the underlying hash function of HMAC. In order to launch the above key-recovery attacks on TUAK, one is first required to find a good differential path for the Keccak permutation. For the f_1 algorithm with key of M bits, one must obtain a differential path with probability less than 2^{-M} to launch a differential-style key-recovery attack. Similarly, for $f_2 \sim f_5$ and f_5^* with key size M bits, one also requires a good differential path with probability less than 2^{-M}. In [18], Daemen and van Assche showed that the lower bound of the probability of a differential path for the Keccak permutation is 2^{-296}, which is much lower than 2^{-256}. To the TUAK algorithm set, the above differential-style key recovery attack cannot be mounted efficiently.

4.7 Extension attacks

An extension attack on a hash function or MAC is a generic attack as it leads to an existential forgery attack to some hash functions. In an extension attack, an attacker first searches for a pair of messages (M, M') colliding at the last l bit internal state of the underlying hash function and then an additional message block X is appended to both messages M and M', i.e., $M \| padding(M) \| X$ and $M' \| padding(M') \| X$ are constructed. Now these two new messages produce the same output as their last l bit of the internal state are the same. A number of attacks on HMAC can be found in [49, 50, 37]. An extension attack may also help to mount a distinguishing attack and a key recovery attack [25, 15]. For a MAC forgery attack, the complexity of forging a MAC using the extension attack is about $2^{\frac{l}{2}}$.

We argue that why the f_1 algorithm is not susceptible to an extension attack and we do not consider other TUAK algorithms. First of all the construction of f_1 is different from the construction of a HMAC algorithm. The f_1 algorithm produces a MAC over the message RAND, AMF, and SQN, and the length of the message is 192 bits that is fixed. It does not accept an input message of length 192 bits. Since f_1 algorithm is based upon the Keccak permutation, any two messages of length 192 bits never produce an internal collision, they may only produce a collision in the output of f_1. Therefore, a similar extension attack cannot be applied to the f_1 algorithm because of unable to extend the message in the input of TUAK algorithms and of unable to produce an internal collision. The designers of Keccak also mentioned that Keccak does not have the length extension weakness [6].

4.8 Summary

In this chapter, we analyzed the $f_1 \sim f_5$, f_1^* and f_5^* algorithms by considering interpolation attacks, algebraic attack, cube attack, slide attacks, differential-type key recovery attacks, and extension attacks. Our analysis shows that the TUAK algorithm set is resistant to these attack. The time complexity of the key recovery attack is not better than the exhaustive key search. The extension attack cannot be applied to the TUAK algorithms due to the input assignment of TUAK.

Chapter 5

Security Proof of TUAK

In this chapter, we present the security proofs of the TUAK construction. The f_1 and f_1^* functions are two MAC algorithms, and the f_2 algorithm is used for generating responses. The f_3, f_4, f_5 and f_5^* functions are key derivation functions. We reduce the security of the function f_i to one or more properties of the sponge function, and then use the known results of the sponge function to deduce the security bound of the function f_i.

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 little bit special. It is used to generate the response in the authentication process. However, it is essentially a kind of MAC. The threat model of f_2 is that the adversary knows everything except the key, and she tries to either recover the key or forge a response. Thus, f_2 should resist to all the attacks applied to MAC. This section investigates the properties of these three functions as MAC algorithms. At the beginning, we take f_1 as an example to derive the security bound. After that, we generalize the discussion on f_1 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 $\underbrace{1\cdots 1}_{5}\underbrace{0\cdots 0}_{314}1$ are the *Sakura* padding [52]. 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 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.

Lemma 1. *Let u and v be two constants, and $u, v \to \infty$, $|u - v| >> 0$. The function $g(x)$ is defined by*

$$g(x) = x^{u-x}\frac{u!}{(u-x)!}\begin{pmatrix} v \\ x \end{pmatrix}, \; x \le \min(u,v).$$

When $x \le \delta$, $g(x)$ is monotonically increasing, where δ is $\min(u,v)$, *i.e.*

$$\delta = \begin{cases} u, \; u << v; \\ v, \; v << u. \end{cases}$$

Proof. We only prove the case that $u << v$. The argument for $v << u$ is the same. We omit here. Let us first consider

$$\frac{g(x)}{g(x+1)} = \frac{x^{u-x}\frac{u!}{(u-x)!}\begin{pmatrix} v \\ x \end{pmatrix}}{(x+1)^{u-x-1}\frac{u!}{(u-x-1)!}\begin{pmatrix} v \\ x+1 \end{pmatrix}}. \tag{5.1}$$

Equation 5.1 equals

$$\begin{aligned} & \frac{x^{u-x}u!v!(u-x-1)!(v-x-1)!(x+1)!}{(x+1)^{u-x-1}u!v!(u-x)!(v-x)!x!}; \\ = & \frac{x^{u-x}(x+1)^2}{(x+1)^{u-x}(u-x)(v-x)}; \\ = & \left(\frac{x}{x+1}\right)^{u-x}\frac{(x+1)^2}{(u-x)(v-x)}. \end{aligned}$$

Notice $(x/(x+1))^{u-x}$ is always less than 1. Let us consider

$$(x+1)^2-(u-x)(v-x)=(u+v+2)x-uv+1.$$

If $x \le (uv-1)/(u+v+2)$, then $(x+1)^2/(u-x)(v-x) \le 1$. Let $\delta=(uv-1)/(u+v+2) \approx u$, when $u,v \to \infty$, and $u << v$.

Lemma 1 is proven. □

Lemma 2. *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.*

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

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 1, $|\Gamma|=2^n$ 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

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

Lemma 2 is proven. □

Theorem 2. *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*

$$Pr[f_1(x_0) = f_1(x_1)] = \begin{cases} \frac{1}{2^n} & \text{if } n = 32, 64, 128, \\ 0 & \text{if } n = 256. \end{cases}$$

Proof. First, let us consider the case that $n = 32, 64, 128$. By Lemma 1, the size of image set of f_1 is also 2^n with high probability. By Lemma 2, 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

$$Pr[f_1(x_0) = f_1(x_1)] = \sum_{i=0}^{2^n-1} Pr[f_1(x_0) = y_i, f_1(x_1) = y_i], y_i \in \mathbb{F}_2^n \text{ and } y_i \neq y_j \text{ for } i \neq j.$$

Therefore,

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

Then, let us consider the case that $n = 256$. Notice that $|\Omega| = 2^{192}$ in this case. It is far less than 2^n. By Lemma 1, $|\Gamma| = |\Omega|$. The probability of collision is zero.

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 $\mathcal{O}_a$, the attacker can build a game $\mathcal{G}$ to find the pre-image of f_1'. We denote l as the number of queries that the attacker needs to find the pre-image of f_1' using any method she wants except the game $\mathcal{G}$. The notation b is the lower bound of the expected number of queries to find the pre-image of f_1' using any method.

Theorem 3. *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.

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.

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

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.

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 if 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.

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. $\square$

By Bertoni et al. [6], the value of l is bounded by $2^{n-r} + 2^{c/2}$. Because

$|\Gamma|/2^m \approx 1$, the result of Theorem 3 becomes

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

Since b is the lower bound of all cases, $b < 2^k$. But as long as b is not far smaller than 2^k and N is negligible compared with 2^k, the value of $2^k - b/2^k - N$ is negligible.

Corollary 2. *As long as $2^k - b << 2^k$, there is no universal key recovery attack on f_1 with N queries, where $N << 2^k$.*

Theorem 4. *An attacker is allowed to select N messages adaptively and access the oracle $\mathcal{O}_t^K$ to get the MAC of each query. After N queries, the attacker is asked to forge the MAC of a message other than the previously queried messages. 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. [6], when using a sponge function as a MAC algorithm, the probability 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

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

This completes the proof. □

Corollary 3. *It is infeasible to forge a MAC generated by f_1.*

The workload in Theorem 4 is greater than 2^{256}, which is the workload of guessing the key. Thus, we consider f_1 resists to the forgery attack as a MAC algorithm.

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^*. Therefore, 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 3 and its corollary also holds for f_2. Theorem 4 needs some tiny modification.

Theorem 5. *An attacker is allowed to select N messages adaptively and access the oracle $\mathcal{O}_t^K$ to get the MAC of each query. After N queries, the attacker is asked to forge the MAC of a message other than the previously queried messages. The expected number of queries to successfully forge a MAC generated by f_2 is*

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

From the proof of Theorem 4, and the cardinality of the pre-image set of f_2, Theorem 5 is straightforward. Corollary 3 still holds because this workload is also greater than guessing the key.

5.2 Security proof of the f_3 - f_5 and f_5^* function construction

The f_3 - f_5 functions are key derivation functions (KDF). The basic requirement of a KDF is that given the random number, the adversary should not have the ability to guess the derived key. For two different random numbers, the f_3 - f_5 should not produce the same key. As a KDF, it is also required to protect the long term credential. In other words, given the derived key, the adversary should not have the ability to guess the original key.

Remark 1. *One observation is that f_2 - f_5 share one function. To be as secure as a random oracle, the sponge function always requires the output length smaller then half of the capacity. However, in this case, the total length of the output of f_2 - f_5 lies between 336 and 816. Even we consider the effective capacity of TUAK is 768, 816 is still far greater than half of 768. Although we cannot find any immediate attacks, this construction has some potential risk.*

We suggest to use different $INSTANCE$ values for f_2 - f_5. This may compromise the efficiency, but the gain is that the construction follows the security assumption. 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 illustrate, define f_i' for $3 \le i \le 5$ the same way as f_1'.

Theorem 6. *Randomly pick x_0 and x_1 from the pre-image set of f_i ($3 \le i \le 5$). The probability of finding a collision of f_i is given by*

$$Pr[f_i(x_0) = f_i(x_1)] = \begin{cases} \frac{1}{2^n} & \text{if } n = 32, 64, 128, \\ 0 & \text{if } n = 256. \end{cases}$$

Theorem 6 implies that the probability that f_i ($3 \le i \le 5$) generates same key for different inputs is negligible.

Theorem 7. *The expected success rate of the universal key recovery attack with N queries under the adaptive chosen plaintext attack on f_i ($3 \le i \le 5$) 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_i.

This theorem means it is hard to recover the long term credential.

Theorem 8. *An attacker is allowed to select N inputs adaptively and access the oracle $\mathcal{O}_{t_i}^{K}$ to get the derived key of each input. After N queries, the attacker is asked to generate a derived key other than the previously queried derived keys. 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 8 shows that it is impossible to forge a derived key generated by f_i $(3 \leq i \leq 5)$ without knowing the long term key. The f_5^* function is the same as the revised f_5, only the constant is different. 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. The function f_1 and f_1^* as two MACs are secure in the sense of collision, key recovery, and forgery. The constructions of f_2 - f_5 are not in a recommended way. We fixed the construction and proved they are secure KDF in the sense of collision, derived key recovery, and original key recovery.

Chapter 6

Cryptanalysis of TUAK in the Multi-output Filtering Model: Part I

In the following two Chapters, we will develop a new type of cryptanalytic tool, called *multi-output filtering model*. As mentioned in Chapter 1, this technique belongs to a general notion called subset cryptanalysis and it is the generalization of the classical filtering model. We split the content into two Chapters. In this Chapter, we give necessary background regarding this new model and develop a general distinguishing attack. In particular, we make use of the cryptographic properties of the associated sequences of the primitive $\mathcal{C}$ under the multi-output model. In the next Chapter, properties of the associated Boolean functions will be used to analyze TUAK.

6.1 Background of multi-output filtering model

Let $\mathcal{C}$ be a cryptographic scheme (keyed or non-keyed) with n-bit input and m-bit output. Clearly it can be simply regarded as a vectorial Boolean function from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. When $\mathcal{C}$ involves a key K, we should write C_K for strictness, but we prefer to use $\mathcal{C}$ for simplicity if the context is clear. In most circumstances, the cryptographic properties of $\mathcal{C}$, such as algebraic degree and nonlinearity, are difficult to be exploited due to the large values of n and m. A natural idea to overcome this difficulty is to restrict the inputs of $\mathcal{C}$ on a subspace $\mathcal{S}$ of $\mathbb{F}_2^n$. For instance, the subspace $\mathcal{S}$ can be generated by an ℓ-stage linear feedback shift register (LFSR). Then we obtain a function $\mathcal{C}'$ from $\mathcal{S}$ to its image set $\mathcal{C}(\mathcal{S})$. By adapting the size of $\mathcal{S}$, we can study the cryptographic properties of $\mathcal{C}'$. If $\mathcal{C}$ has good randomness properties, it should be difficult to find a subspace $\mathcal{S}$ such that $\mathcal{C}'$ has bad randomness properties. We must mention that the above method for analyzing the cryptographic scheme $\mathcal{C}$ lies in a more general notion called *subset cryptanalysis* [35], which tries to track the statistical evolution of a certain subset of values through various operations in the cryptographic schemes. One is referred to [21] for a success application of the subset cryptanalysis to find 5-round collision on Keccak.

We achieve the above idea by proposing a new technique, called a *multi-output filtering model*. This model aims to exploit the non-randomness property of a cryptographic algorithm $\mathcal{C}$ such as message authentication codes and block ciphers. General speaking, a multi-output filtering model consists of a linear feedback shift register and a multi-output filtering function. The LFSR is used to generate an input subspace of $\mathcal{C}$ and $\mathcal{C}$ is used a multi-output filtering function. This multi-output model is a generalization of the classic filtering model in stream ciphers [46] as it outputs multiple bits, instead of only one bit, for the set of inputs to $\mathcal{C}$ generated by an LFSR. Under this model, we can obtain a number of component sequences and component functions in the multi-output model. This paper is devoted to studying the randomness properties of $\mathcal{C}$ through investigating its component sequences and component functions. We should mention that in this paper we restrict $\mathcal{C}$ to MACs and block ciphers, but this model can also be generalized to study other cryptographic primitives.

6.2 Preliminaries

In this section, we provide some definitions and results that will be used in this paper.

6.2.1 Basic definitions on sequences

We start with presenting some definitions on sequences. For a well-rounded treatment of sequences and Boolean functions, the reader is referred to [14, 27].

Let $\mathbf{s} = \{s_i\}$ be a sequence generated by a linear feedback shift register (LFSR) whose recurrence relation is defined as

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

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 $\mathbb{S}$ in Eq. (6.1) with period $2^n - 1$ generated by an LFSR is called an *m-sequence*. Let $\mathbf{s} = \{s_i\}$ be an m-sequence of period $2^\ell - 1$ and $f(x_0, \ldots, x_{\ell-1})$ be a Boolean function in ℓ variables. We define a sequence $\mathbf{a} = \{a_i\}$ as

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

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 [27]. On the other hand, the linear complexity of a nonlinear filtering sequence lies in the range of ℓ and $2^\ell - 1$ [32]. If a filtering sequence has linear complexity $2^\ell - 1$, then we call it has *optimal* linear complexity.

6.2.2 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 [27].

Definition 1. *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 [39].

Definition 2. *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 $\mathrm{NL}(f) \geq 2^{n-1} - 2^{n/2-1}$ [14]. 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 a (n,m)-function, multi-output Boolean functions, or vectorial Boolean functions, where f_i's are called coordinate functions [14].

6.3 Multi-Output Filtering Model

In this section, we provide a detailed description of the multi-output filtering model of a cryptographic primitive.

6.3.1 Description of the multi-output filtering model

Let $\mathbf{a} = \{a_i\}_{i\geq 0}$ be a binary sequence generated by an ℓ-stage linear feedback shift register (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 as follows $\mathcal{R} = \{\mathrm{R}_j : 0 \leq j \leq 2^\ell - 2\}$ where

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

where 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 multi-output filtering model 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

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

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, $0 \le i \le m-1$.

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

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

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 $g_i : \mathbb{F}_{2^\ell} \rightarrow \mathbb{F}_2$ $(0 \le i \le m-1)$ are defined as

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

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

6.3.2 Application to TUAK's f_1, AES, KASUMI and PRESENT

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

6.3.2.1 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 $\text{STATE}_i \in \mathbb{F}_2^{\ell}$. We obtain $2^{\ell} - 1$ different T-bit RAND numbers $\mathcal{R} = \{\text{RAND}_j : 0 \leq j \leq 2^{\ell} - 2\}$ by Eq. (6.3) and the component sequences and component functions are obtained using Eq. (6.4) with $\text{C}_j = f_1\left(\text{K}, \text{R}_j, \text{SQN}\right)$.

Remark 2. *For TUAK's f_1 function, in Eq. (6.5), recovering the last bit $y_{2^{\ell}-2,i}$ for each component sequence $\mathbf{s}_i$ 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.*

6.3.2.2 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 [22]. 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 $\text{C}_j = \text{AES\_128}\left(\text{K}, \text{R}_j\right)$ and $\text{C}_j = \text{AES\_256}\left(\text{K}, \text{R}_j\right)$. PRESENT [9] 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 $\text{C}_j = \text{PRESENT}\left(\text{K}, \text{R}_j\right)$. KASUMI [53] 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 $\text{C}_j = \text{KASUMI}\left(\text{K}, \text{R}_j\right)$.

6.4 Distinguishing Attack Model IND-CPA

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 [26] in public-key settings. In [4], 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 multi-output filtering model 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 random oracle by $\mathcal{O}$ and the adversary by $\mathcal{A}$. The indistinguishability game [3, 26] between the random oracle and the adversary is played 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 random 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.

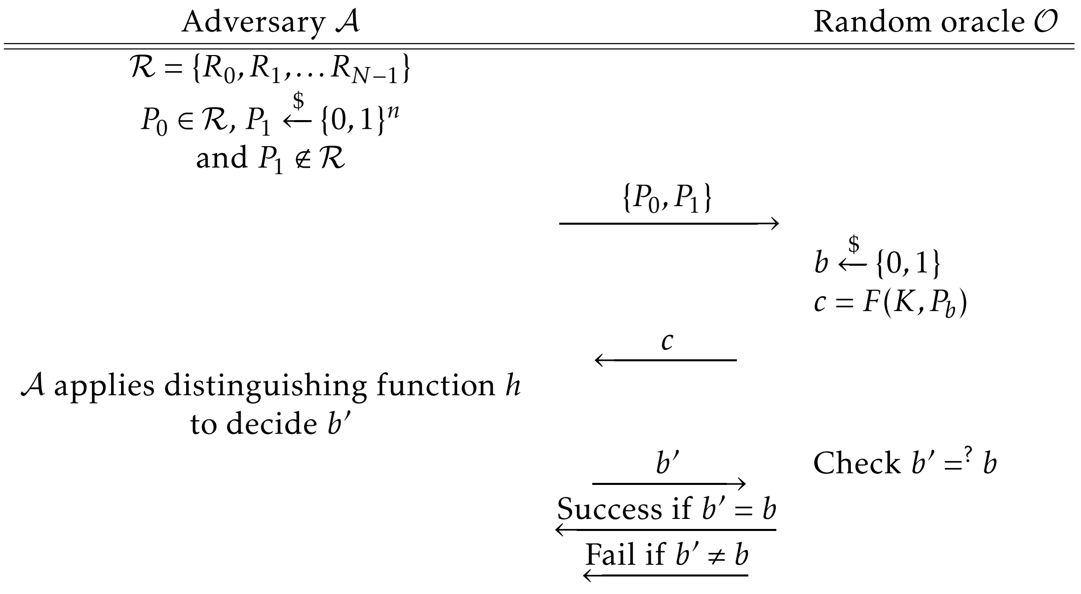
Adversary $\mathcal{A}$
Random oracle $\mathcal{O}$
$\mathcal{R} = \{R_0, R_1, \ldots R_{N-1}\}$
$P_0 \in \mathcal{R}$, $P_1 \overset{\$}{\leftarrow} \{0,1\}^n$
and $P_1 \notin \mathcal{R}$
$\{P_0, P_1\}$
$b \overset{\$}{\leftarrow} \{0,1\}$
$c = F(K, P_b)$
c
$\mathcal{A}$ applies distinguishing function 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.

6.5 A Generic Framework to Build a Distinguisher Under IND-CPA Model

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. We start with the following definition.

Definition 3. *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*

$$\sum_{i \in \{0,1\}} \Pr\Big(h(C(P_i)) = i\Big) \cdot \Pr\Big(\text{oracle chooses } i\Big) \tag{6.7}$$

is non-negligible compared with $1/2$, when (P_0, P_1) is randomly chosen from Ω. By assuming $\Pr\Big(\text{oracle chooses } i\Big) = 1/2$, Eq. (6.7) can be simplified as saying

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

is non-negligible compared with 1.

Now we state the main theorem below and provide the proof of it in Appendix B due to the page limit.

Theorem 9. *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*

$$h(y) = \begin{cases} 0 & \text{if } y = \mathcal{C}(x) \text{ and } x \in \mathcal{CS}, \\ 1 & \text{otherwise.} \end{cases} \tag{6.8}$$

Define the following two probabilities

$$\begin{aligned} q_0 &= \Pr(x_0 \in \mathcal{R} \wedge x_0 \in \mathcal{CS}), \\ q_1 &= \Pr(x_1 \in \mathcal{S}' \wedge x_1 \in \mathcal{CS}). \end{aligned} \tag{6.9}$$

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

$$\sum_{i \in \{0,1\}} \Pr\Big(h(C(P_i)) = i\Big) \cdot \Pr\Big(\textit{oracle chooses } i\Big) = \frac{1 + |q_0 - q_1|}{2}. \tag{6.10}$$

Proof. Let c be the ciphertext sent back from the oracle. 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 $i \in \{0, 1\}$, theorefore 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

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

(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

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

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

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

Summarizing the above four probability values we have the desired result that

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

The proof is completed. □

Several remarks on Theorem 9 are as follows:

(i) One may wonder how $h(y)$ in Eq. (6.8) can be computed and how it can be computed efficiently. This is a natural question as we could not decide whether $x = \mathcal{C}(y)$ (it will imply $y = \mathcal{C}^{-1}(x)$ but y is unknown to the attacker). Also, if the set $\mathcal{CS}$ is very large, one cannot determine the whole image set $\mathrm{Im}_{\mathcal{C}}(\mathcal{CS})$ and then tell whether y is in it or not. To clear this doubt, it is important to mention that we define the set of CS according to some property of the cipher. Therefore when one tells whether $y \in \mathcal{C}(\mathcal{CS})$ he does not need to know the whole image set but only tell whether y satisfies the property or not. One may refer to the next Section for an example of such condition set.

(ii) An attacker will expect the probability value in (6.10) to be as large as possible so that she can distinguish the cryptographic scheme $\mathcal{C}$ with a high probability.

(iii) The difficulty of finding the distinguishing attack described in Theorem 9 is to find a proper condition set $\mathcal{CS}$ such that $|q_0 - q_1|$ is large. 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.

6.6 Distinguishing Attack Based on Linear Complexity

In this section, 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.

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 [41]. 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 [41] 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.

6.6.1 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 $P_0 \xleftarrow{\$} \mathcal{LI}$ and $P_1 \xleftarrow{\$} \overline{\mathcal{LI}}$. 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

$$\frac{\phi(2^\ell - 1)}{\ell} \times N_{\text{key}} \times 2\ell \times (2^\ell - 1) \times m, \tag{6.11}$$

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.

6.6.2 Distribution of f_1, AES, KASUMI and PRESENT.

In our experiment, we choose $\ell = 8$ and $N_{key} = 10^8$. By Eq. (6.11), 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).

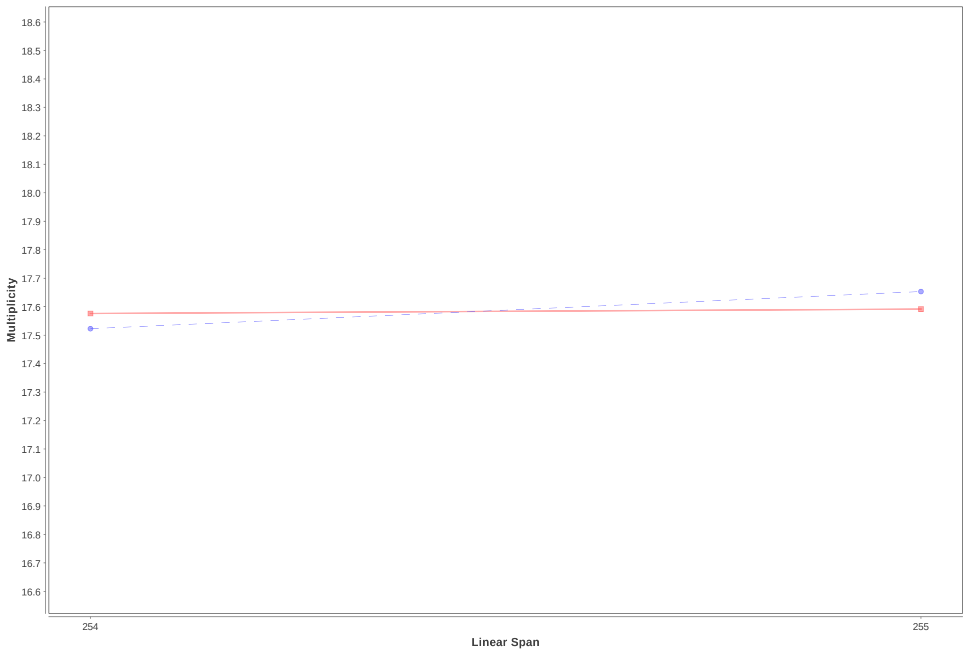

Figure 6.2: Kasumi

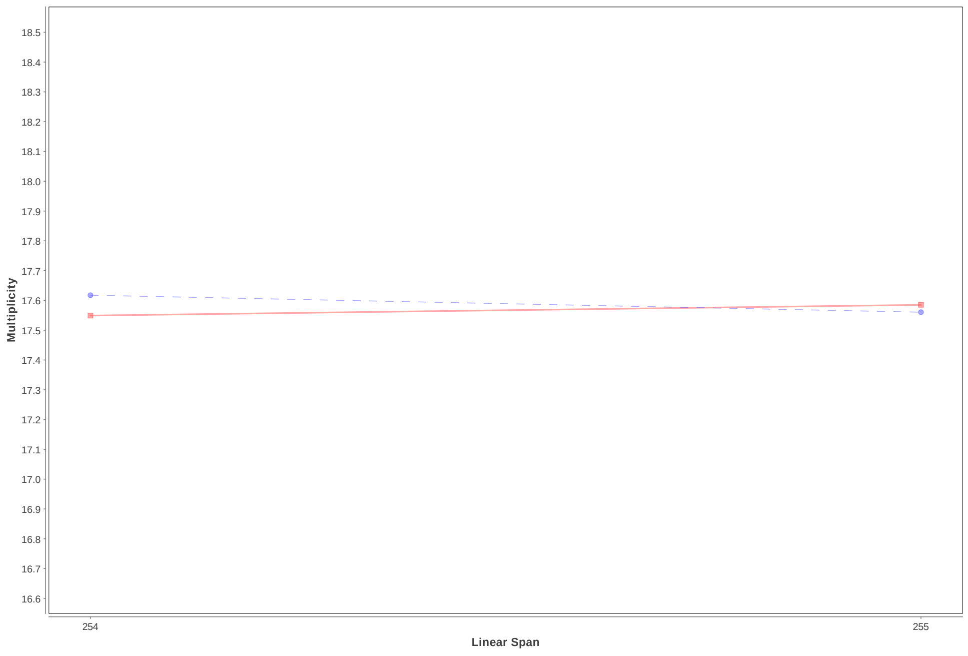

Figure 6.3: Present

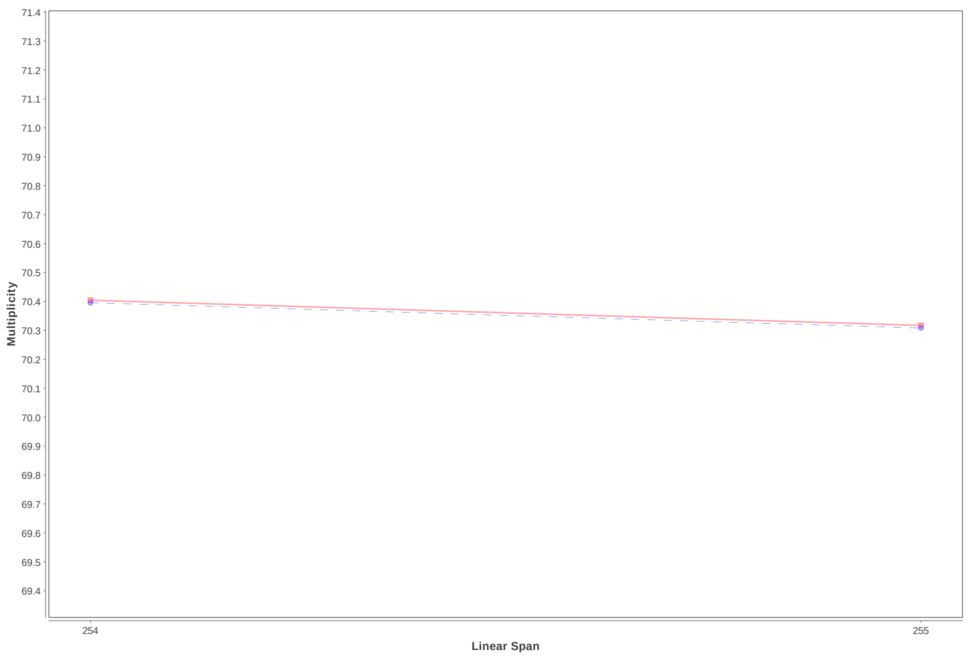

Figure 6.4: AES

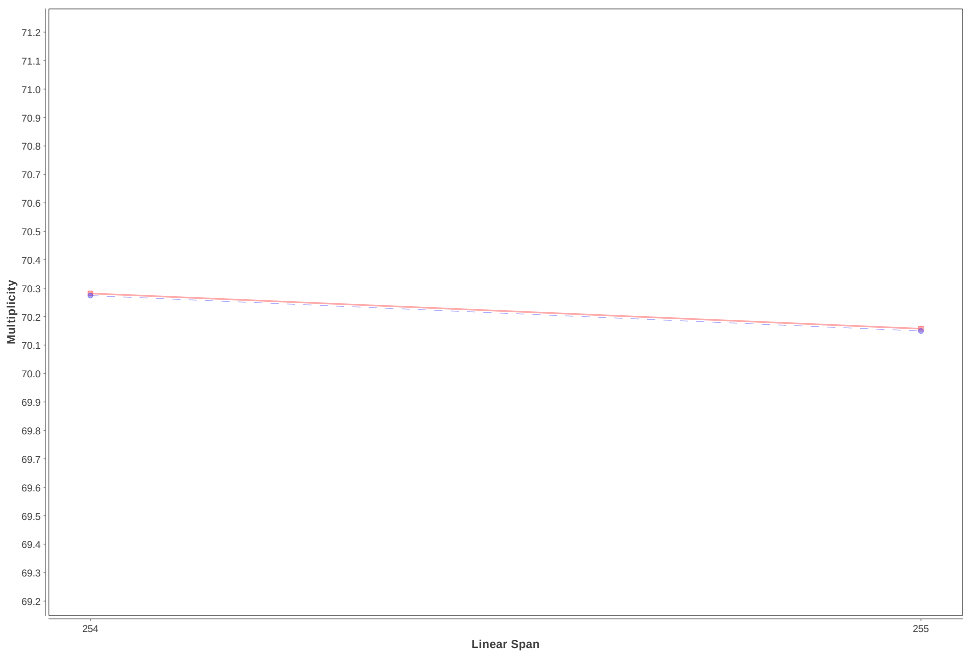

Figure 6.5: f_1

From Figs. 6.2 and 6.3, 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 Figs. 6.4 and 6.5 show this is not the case for AES and f_1.

6.6.3 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 based on the linear complexity distribution of the component sequences of a primitive in the multi-output filtering model.

6.6.3.1 Constructing the distinguishing function.

Recall that the distinguishing function is defined in Definition 3. We use the notations in Theorem 9 and the attack model is depicted in Fig. 6.1.

1. Choosing an ℓ-stage LFSR with a primitive polynomial to generate the inputs of length m 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 $\ell \le LC \le 2^\ell - 1$;

5. Defining the condition set

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

6. The distinguishing function h is defined in Eq. (6.8) using the condition set $\mathcal{CS}$;

7. q_0, q_1 are the probability values defined in Definition 3.

6.6.4 An example of the attack

In this section, we apply the attack with our distinguishing function defined in Section 6.6.3 on f_1, AES, KASUMI, and PRESENT. Theorem 9 and the observations in Figs. 6.2 and 6.3 enable us to gain a non-negligible success rate of the attack on 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 the computer experiments we performed below, we are testing the success rate of our attack by testing more data. The details of the results will be published in a revised version later.

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 Fig. 6.1, we use the distinguishing function h to execute the attack. It is worth to mention that, to test 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 all experiments. We present the average success rate for an experiment in Table 6.1, where we use the upper bound of the slope t and the 8-stage LFSR the same as those in Table 6.2 below.

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%

The parameters we choose in the above experiment is as follows. The slope in Table 6.2 is the average slope over 10^8 samples. The column

"Slope (L)" contains the slopes computed from the LFSR input, and the column "Slope (R)" contains the slopes computed from the random input. The last column shows the absolute value of the difference between "Slope(L)" and "Slope(R)". We can see the "Difference" of KASUMI and PRESENT are much greater than f_1 and AES.

Table 6.2: The slope of f_1, AES, KASUMI and PRESENT on average

Primitive	Polynomial of the LFSR	Slope (L)	Slope (R)	| Difference |
f_1	$x^8+x^6+x^4+x^3+x^2+x^1+1$	-0.125	-0.124	2.210×10^{-4}
AES	$x^8+x^7+x^6+x^3+x^2+x^1+1$	-0.088	-0.087	5.190×10^{-4}
KASUMI	$x^8+x^7+x^6+x^5+x^4+x^2+1$	0.130	0.015	0.115
PRESENT	$x^8+x^6+x^5+x^3+1$	-0.057	0.036	0.093

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.

6.7 Summary

In this section, we elaborate the new cryptanalytic technique multi-output filtering model. The application of this model on a cryptographic primitive may lead to the distinguishing attack of it under IND-CPA model if the primitive is designed with certain flaw. As the applications, we test the performance of f_1, AES, KASUMI and PRESENT under this attack. We found that, as shown in Table 6.1, f_1 performs very like AES, both of which have excellent randomness properties can this attack cannot mount on them. However, the non-negligible average success rate of this attack on KASUMI and PRESENT shows that this attack is not a trivial one.

Chapter 7

Cryptanalysis of TUAK in the Multi-output Filtering Model: Part II

In the previous Chapter, we study the random property of TUAK functions by considering the linear complexities of the sequences. In this Chapter, we will evaluate the randomness of TUAK functions by investigating the other cryptographic properties of the component functions (defined Section 6.3.1). The properties we will discuss in this Chapter are the algebraic degree and the nonlinearity of Boolean functions. The good performance of the component functions on these criteria would suggest the good random property of TUAK functions.

7.1 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 multi-output filtering model. 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.

7.1.1 Algebraic degree distribution

Recall that the algebraic degree of a Boolean function is defined in Section 6.2. 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 [14]. We provide a simple proof below for the completeness.

Theorem 10. *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 $\frac{2^{\sum_{i=0}^{d}\binom{n}{i}}}{2^{2^n}}$. In particular, the probability that the algebraic degree is not more than $n-2$ is $\frac{1}{2^{n+1}}$.*

Several remarks on the application of Theorem 10 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 [40] 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 $\mathcal{C}_F(x) = \sum_{I \in \mathcal{P}(\Omega)} a_I(K) x^I$, 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) [14]. Then the restrictions of $\mathcal{C}_F|_{\mathcal{S}} = \sum_{I \in \mathcal{P}(\Omega), I \subset \mathcal{S}} a_I(K) x^I$. 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 10 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 multi-output filtering model as in Section 6.3.1. 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 7.1: Distribution of the degree smaller than 7

Cryptographic primitive	$\Pr(d \leq 6)$
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 7.1, we can see that for KASUMI*, the probability* $Pr(d \leq 6)$ *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 7.1.*

7.1.2 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 6.2. 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 $\left(f(x)+g(x) : x \in \mathbb{F}_2^n\right)$. 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 [14] 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 11 ([44, 14]). *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 7.2: 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	0.54747879061478987802997919600 8
97	0.71902310151075402984781111703 1
96	0.82824221024964787460062882576 5
95	0.89663430607249953273680824529 9
94	0.93875783156791138635120360571 3
93	0.96427774651450020080734327382 1
92	0.97948642802537161847775244745 5
91	0.98840267349024055409268340534 3
90	0.99354511316750952827725848552 4

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

$$1 - 0.993545113167509528277258485524 \approx 0.00645.$$

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

Cryptographic primitive	$\Pr_{<90}$	$\Pr_{<91}$
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).

7.2 Summary

In this Chapter we study the distribution of the algebraic degrees, nonlinearities of the component functions of the primitives f_1, AES, KASUMI and PRESENT. We first determine these two distributions for using a random primitive as the multi-output filter, and then compare it with those for the above four primitives. Although we cannot find an attack by making use of these properties, some nonrandomness properties can still be observed from our experiments. For instance, one may see that there are more component functions of KASUMI having degree less than $r-2$, which is a potential risk as an efficient decoding method of the Reed-Muller codes may yield an attack on it. However, for f_1, it has very good randomness properties just like AES.

Chapter 8

Concluding Remarks

The TUAK algorithm set contains authentication and key generation functions that provide security functionalities in the 3GPP system. Among the algorithms in TUAK, f_1 and f_1^* are treated as message authentication functions and f_2 to f_5 (f_5^*) are treated as key derivation functions. To study the security of each algorithm in TUAK, we first apply various known attacks on it and show the performance of it under those attacks. By presenting the complexity of these attacks on each algorithm, we conclude that the algorithms in TUAK are resistant to them.

Next, by constructing an adversary using an oracle, we reduce the security of TUAK to the security of Keccak. Theoretically, we show that f_1 and f_1^* are immune to the key recovery attack and universal forgery attack under the chosen plaintext attack; and other algorithms are immune to the key recovery and collision attack under the chosen plaintext attack.

Finally, we develop a new type cryptanalytic tool, called multi-output filtering model, to test the indistinguishability of a cryptographic primitive under the so-called IND-CPA model. We provide a generic method to build a distinguisher for a cryptographic primitive. In particular, based on the distribution of the linear complexity of the component sequences, we propose a new distinguishing attack method. As applications, the randomness properties of TUAK, AES, KASUMI and PRESENT are studied under this newly developed technique. Note that the reason for testing the other three primitives is to obtain a better comparison to the performance of TUAK under this attack. By studying the component sequences and component Boolean functions of the above primitives, we show that TUAK and AES are properly designed under the multi-output model, while a distinguishing attack can be mounted on KASUMI and PRESENT with a non-negligible average success rate.

In conclusion, through our extensive analysis of the TUAK algorithm set, we believe that the algorithms in TUAK perfectly inherit the good security performance from Keccak and it can be used with confidence as message authentication functions and key derivation functions.

References

[1] J.-P. Aumasson and Meier W., “Zero-sum Distinguishers for Reduced Keccak-f and for the Core Functions of Luffa and Hamsi”, Presented at the rump session of Cryptographic Hardware and Embedded Systems - CHES 2009, 2009.

[2] J.-P. Aumasson, I. Dinur, W. Meier, and A. Shamir, “Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium”, *Fast Software Encryption*, LNCS 5665, pp 1 – 22, 2009.

[3] Bellare, M., Desai, A., Pointcheval, D., Rogaway, P.: Relations among notions of security for public-key encryption schemes. Advances in Cryptology âĂŤ CRYPTO ’98, LNCS, vol. 1462, pp. 26 – 45. Springer Berlin Heidelberg (1998)

[4] Bellare, M., Desai, A., Jokipii E., Rogaway, P.: A concrete security treatment of symmetric encryption: Analysis of the DES modes of operation. Proceedings of the 38th Symposium on Foundations of Computer Science, IEEE (1997)

[5] G. Bertoni, J. Daemen, M. Peeters, G.V. Assche, “The Keccak Reference” `http://keccak.noekeon.org/Keccak-reference-3.0.pdf`, 2011.

[6] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, “Cryptographic Sponge Functions”, January 2011, http://sponge.noekeon.org/.

[7] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak Sponge Function Family Main Document", Submission to NIST (updated), Version 1.2, 2009.

[8] E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems", Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology, pp. 2 – 21, 1991.

[9] Bogdanov, A., Knudsen, L.R., Leander, G. Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An ultra-lightweight block cipher, Cryptographic Hardware and Embedded Systems - CHES 2007, LNCS, vol. 4727, pp. 450 – 466. Springer Berlin Heidelberg (2007)

[10] A. Biryukov and D. Wagner, "Slide Attacks", 6th International Workshop on Fast Software Encryption (FSE '99), LNCS, pp. 245 – 259, Springer-Verlag, March 1999.

[11] A. Biryukov and A. Shamir, "Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers", Advances in Cryptology-Asiacrypt 2000, LNCS 1976, pp. 1–13, Springer-Verlag, 2000.

[12] C. Boura, and A. Canteaut, "Zero-sum Distinguishers for Iterated Permutations and Application to Keccak-f and Hamsi-256", In: A. Biryukov, G. Gong, D.R. Stinson (eds.) SAC 2011, LNCS 6544, pp. 1 – 17, Springer-Heidelberg, 2011.

[13] C. Boura, A. Canteaut and C. De Canniére, "Higher-Order Differential Properties of Keccak and Luffa", Fast Software Encryption, LNCS 6733, pp. 252 – 269, 2011.

[14] C. Carlet, Boolean Functions for Cryptography and Error Correcting Codes, Chapter of the monography "Boolean Models and Methods in Mathematics, Computer Science, and Engineering", Cambridge University Press, Yves Crama and Peter L. Hammer (eds.), pp. 257-397, 2010

[15] S. Contini and Y.L. Yin, "Forgery and Partial Key-Recovery Attacks on HMAC and NMAC Using Hash Collisions", In Lai, X., Chen, K., eds.: ASIACYPT, LNCS 4284, Springer, 2006.

[16] N. Courtois and W. Meier, "Algebraic Attacks on Stream Ciphers with Linear Feedback", Advances in Cryptology-Eurocrypt'03, LNCS 2656, pp. 345 – 359, Springer-Verlag, 2003.

[17] N. Courtois, A. Klimov, J. Klimov, and A. Shamir, "Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations", In B. Preneel, (ed.), Proceedings of Eurocrypt 2000, LNCS 1807, pp. 392 – 407, Springer-Verlag, 2000.

[18] J. Daemen and G. Van Assche, "Differential Propagation Analysis of Keccak", Fast Software Encryption FSE 2012, pp. 422 – 441, 2012.

[19] I. Dinur, and A. Shamir, "Cube Attacks on Tweakable Black Box Polynomials", Advances in Cryptology-EUROCRYPT '09, LNCS, pp. 278–299, Springer-Verlag, 2009.

[20] I. Dinur, O. Dunkelman, and A. Shamir, "Collision Attacks on Up to 5 Rounds of SHA-3 Using Generalized Internal Differentials", Fast Software Encryption, 2013.

[21] Dinur, I., Dunkelman, O., Shamir, A., Collision attacks on up to 5 rounds of SHA-3 using generalized internal differentials. Cryptology ePrint Archive, Report 2012/627. (2012) http://eprint.iacr.org/

[22] Daemen, J., Rijmen, V.: The Design of Rijndael, AES – The Advanced Encryption Standard. Springer (2002)

[23] I. Dinur, O. Dunkelman, and A. Shamir, "New Attacks on Keccak-224 and Keccak-256", Fast Software Encryption 2012, LNCS 7549, pp. 442 – 461, Springer Berlin Heidelberg, 2012.

[24] I. Dinur, P. Morawiecki, J. Pieprzyk, M. Srebrny and M. Straus. "Practical Complexity Cube Attacks on Round-Reduced Keccak Sponge

Function", Cryptology ePrint Archive, Report 2014/259, 2014. http://eprint.iacr.org/

[25] P.-A. Fouque, G. Leurent, P.Q. Nguyen, "Full Key-Recovery Attacks on HMAC/NMAC-MD4 and NMAC-MD5", Advances in Cryptology - CRYPTO 2007, LNCS 4622, pp. 13 – 30, 2007.

[26] S. Goldwasser and S. Micali, "Probabilistic encryption", Journal of Computer and System Sciences 28, 270 – 299 (1984)

[27] S.W. Golomb and G. Gong, Signal Design for Good Correlation – for Wireless Communication, Cryptography and Radar, Cambridge Press, 2005.

[28] M. Gorski, S. Lucks, T. Peyrin, "Slide Attacks on a Class of Hash Functions", Advances in Cryptology – ASIACRYPT 2008, LNCS 5350, pp. 143 – 160, 2008.

[29] E. Homsirikamol, P. Morawiecki, M. Rogawski, and M. Srebrny, "Security Margin Evaluation of SHA-3 Contest Finalists Through Sat-Based Attacks", In A. Cortesi, N. Chaki, K. Saeed, and S.T. Wierzchon, (eds.), CISIM, LNCS 7564, pp. 56 – 67, Springer, 2012.

[30] T. Jakobsen, L.R. Knudsen, "The Interpolation Attack on Block Ciphers", Fast Software Encryption 1997, LNCS 1267, pp. 28 – 40, 1997.

[31] A. Joux, and T. Peyrin, "Hash Functions and the (Amplified) Boomerang Attack", Advances in Cryptology - CRYPTO 2007, LNCS 4622, pp. 244 – 263, 2007.

[32] Key, E.L.: An analysis of the structure and complexity of nonlinear binary sequence generators. IEEE Transactions on Information Theory 22, 732 – 736. (1976)

[33] J. Kim, A. Biryukov, B. Preneel, S. Hong, "On the Security of HMAC and NMAC based on HAVAL, MD4, MD5, SHA-0 and SHA-1 (ex-

tended abstract)”. In SCN 2006 (2006), Prisco R. D., Yung M., (Eds.), LNCS 4116, pp. 242âĂŞ256, Springer, 2006.

[34] X. Lai, M. Duan, “Improved Zero-sum Distinguisher for Full Round Keccak-f Permutation”, Cryptology ePrint Archive, Report 2011/023 (2011), http://eprint.iacr.org/2011/023

[35] G. Leander, M.A. Abdelraheem, H. AlKhzaimi, E. Zenner, " A crypt-analysis of PRINTcipher: The invariant subspace attack". In Rogaway, P. (ed.), CRYPTO 2011. LNCS, vol. 6841, pp. 206 – 221. Springer (2011)

[36] J. Lathrop, “Cube Attacks on Cryptographic Hash Functions [EB/OL]”, Master’s Thesis, 2009 http://www.cs.rit.edu/~jal6806/thesis/.

[37] E. Lee, D. Chang, J. Kim, J. Sung, and S. Hong, “Second Preimage Attack on 3-Pass HAVAL and Partial Key-Recovery Attacks on HMAC/NMAC-3-Pass HAVAL”, In Kaisa Nyberg, editor, FSE, LNCS 5086, pp. 189 – 206, Springer, 2008.

[38] G. Leurent, and A. Roy, “Boomerang Attacks on Hash Function Using Auxiliary Differentials”, Topics in Cryptology - CT-RSA 2012, LNCS 7178, pp. 215 – 230, 2012.

[39] M. Matsui, and A. Yamagishi, “A New Method for Known Plaintext Attack of FEAL Cipher”, Advances in Cryptology - EUROCRYPT 1992.

[40] MacWilliams, F.J., Sloane, N.J.A.: The theory of error-correcting codes. North-Holland Mathematical Library. (1977)

[41] W. Meidl and H. Niederreiter, “On the Expected Value of the Linear Complexity and the k-Error Linear Complexity of Periodic Sequences”, IEEE Transaction on Information Theory, vol. 48, No. 11, 2817–2825.

[42] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, Handbook of Applied Cryptography, CRC Press, 1997.

[43] M. Naya-Plasencia, A. Röck, and W. Meier, "Practical Analysis of Reduced-Round Keccak", In D.J. Bernstein and S. Chatterjee, (eds.), Progress in Cryptology - INDOCRYPT 2011, LNCS 7107, Springer-Heidelberg, 2011.

[44] D. Olejar, M. Stanek, "On Cryptographic Properties of Random Boolean Functions", Journal of Universal Computer Science, vol 4, No. 8, pp. 705 – 717, 1998.

[45] T. Peyrin, Y. Sasaki, and L. Wang, "Generic Related-Key Attacks for HMAC", Advances in Cryptology – ASIACRYPT 2012, LNCS 7658, pp. 580 – 597, 2012.

[46] R.A. Rueppel, Analysis and Design of Stream Ciphers, Springer-Verlag, Berlin, 1986.

[47] J.V. Uspensky, Introduction to Mathematical Probability, New York McGraw-Hill, 1937.

[48] D. Wagner, "The Boomerang Attack", Fast Software Encryption, LNCS 1636, pp. 156 – 170, 1999.

[49] K. Yasuda, "Multilane HMAC - Security beyond the Birthday Limit", In K. Srinathan, C. Pandu Rangan, and Moti Yung, editors, INDOCRYPT, LNCS 4859, pp. 18 – 32, Springer, 2007.

[50] K. Yasuda, "A Double-Piped Mode of Operation for MACs, PRFs and PROs: Security beyond the Birthday Barrier", In Antoine Joux, editor, EUROCRYPT, LNCS 5479, pp. 242 – 259, Springer, 2009.

[51] A.M. Youssef, G. Gong, "On the Interpolation Attacks on Block Ciphers", Proceedings of the 7th International Workshop on Fast Software Encryption, FSE 2000, pp. 109 – 120, 2000.

[52] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, "Sakura: a flexible coding for tree hashing", April 2013, http://sponge.noekeon.org/.

[53] 3^{rd} generation partnership project, Technical specification group services and system aspects, 3G security, specification of the 3GPP confidentiality and integrity algorithms; Document 2: KASUMI specification, V.3.1.1, 2001.

Appendix

In this Appendix, we give the experimental results mentioned in the document.

I: Distribution of the nonlinearity of the component sequences of f_1 and AES

We first give two tables of the distribution of the nonlinearity of the component sequences of f_1 which is generated by LFSRs with minimal polynomials of degree 8. Note that there are altogether 16 such LFSRs. We number them in the following table. Also, we randomly pick up 30 keys, which are also numbered in the following table. We first give the tables of the LFSRs and Keys we used.

Table 8.1: LFSRs' Index

LFSR Index	Polynomial
0	$x^8 + x^6 + x^5 + x^1 + 1$
1	$x^8 + x^7 + x^3 + x^2 + 1$
2	$x^8 + x^5 + x^3 + x^2 + 1$
3	$x^8 + x^6 + x^4 + x^3 + x^2 + x^1 + 1$
4	$x^8 + x^7 + x^6 + x^1 + 1$
5	$x^8 + x^7 + x^5 + x^3 + 1$
6	$x^8 + x^7 + x^2 + x^1 + 1$
7	$x^8 + x^6 + x^3 + x^2 + 1$
8	$x^8 + x^7 + x^6 + x^3 + x^2 + x^1 + 1$
9	$x^8 + x^7 + x^6 + x^5 + x^4 + x^2 + 1$
10	$x^8 + x^6 + x^5 + x^4 + 1$
11	$x^8 + x^4 + x^3 + x^2 + 1$
12	$x^8 + x^6 + x^5 + x^3 + 1$
13	$x^8 + x^7 + x^6 + x^5 + x^2 + x^1 + 1$
14	$x^8 + x^5 + x^3 + x^1 + 1$
15	$x^8 + x^6 + x^5 + x^2 + 1$

Table 8.2: Keys' Index

Key Index	Key Value
0	0x81 0x8e 0x9a 0x4c 0x65 0x80 0x2e 0x94 0xc3 0x3 0xb 0xa 0x7b 0xc8 0x2f 0x83
1	0xa0 0xe9 0x5d 0x87 0x2a 0xcd 0x70 0x55 0xeb 0x4b 0xf0 0x26 0xc5 0x12 0xe3 0xb6
2	0x19 0x48 0xe5 0x1 0x82 0x71 0x7b 0x4a 0xc4 0x93 0xa5 0x62 0xff 0x97 0xb2 0x27
3	0x29 0xf3 0x58 0x4b 0xba 0x86 0x4 0x57 0xa9 0xc3 0x6f 0xef 0x98 0x12 0x72 0x57
4	0x4c 0x73 0x1b 0x36 0x5f 0x65 0x0 0x9a 0x39 0x6 0xd3 0x3d 0x3c 0x7b 0x37 0x7
5	0x3f 0x92 0xd2 0xd3 0x3e 0xa6 0xa4 0x76 0x4f 0xc4 0x97 0xfd 0xd8 0xb 0x57 0x37
6	0x0 0x59 0x63 0x74 0x63 0x24 0x65 0x8b 0xa 0xa5 0xbf 0x22 0x9a 0x3c 0x66 0x29
7	0xcb 0x10 0xf3 0xa8 0x1c 0xf6 0xf7 0xbb 0xc6 0x94 0x8f 0xda 0xee 0xc6 0x39 0x5e
8	0x1e 0x40 0xe6 0x41 0xf6 0x75 0x51 0x26 0x1f 0xb8 0x8e 0x99 0x81 0xa1 0xe6 0x97
9	0xb6 0xb3 0xe2 0x51 0xbe 0x3b 0xa6 0x2d 0xf4 0x7b 0x80 0xe 0x40 0x8 0xc1 0xd5
10	0x8f 0x71 0xcc 0x28 0x80 0x21 0x6c 0x72 0x60 0x87 0x6b 0x2a 0x59 0x74 0x60 0x59
11	0xe6 0xc4 0xc8 0x56 0x8a 0x40 0x58 0xd4 0xc2 0xc3 0x92 0x1f 0x38 0x9c 0x97 0xa3
12	0x39 0x34 0x3c 0xae 0x69 0xf1 0x5f 0x77 0xb6 0x59 0x7c 0x5e 0x8a 0x7b 0x7b 0x75
13	0x45 0x8a 0xcc 0x40 0xea 0xcc 0xb6 0xb9 0x18 0xb2 0xec 0x97 0x3d 0x49 0x62 0xd0
14	0x6 0xd1 0x5e 0x5d 0x1a 0xaa 0xd2 0x3d 0x7 0x78 0xe9 0xbc 0x7c 0x7f 0xe0 0xf5
15	0xba 0x4f 0x17 0x96 0x45 0xa6 0x68 0xe3 0xdf 0x92 0xb7 0xfe 0xb6 0xd6 0xca 0x65
16	0xdd 0x8f 0x5b 0xbc 0xf9 0x17 0x6c 0xcd 0x3d 0x2a 0xdb 0xcd 0x98 0x47 0x36 0xe0
17	0x2c 0x59 0xd0 0xe0 0x3 0x97 0x15 0x5a 0xfd 0xa9 0x1a 0xda 0xd 0xc 0x79 0x69
18	0xa5 0xb7 0x5b 0x53 0x6f 0xff 0xd6 0x2e 0x3e 0xb8 0x7a 0x17 0x44 0x9d 0x26 0x3f
19	0x85 0xf2 0x20 0xa7 0x8b 0x67 0x65 0x27 0x5c 0x40 0x3e 0xb5 0xa9 0xb3 0x14 0xe4
20	0x48 0x91 0x85 0xab 0xe5 0x29 0xb8 0x68 0xf4 0x6a 0xed 0x24 0xe9 0x47 0x58 0x19
21	0xac 0x60 0x2f 0x71 0x47 0xde 0x2 0x51 0xe2 0x9e 0x4b 0x16 0xf2 0x93 0x46 0xdf
22	0xad 0x65 0x2 0x4b 0xc1 0x5e 0xb8 0x84 0x45 0x87 0x5d 0x7c 0xf0 0xe 0x73 0x76
23	0x89 0xeb 0x24 0xc9 0x9e 0xc3 0x91 0xe0 0x79 0xd 0x68 0x86 0x50 0x73 0xb4 0x60
24	0xbc 0x7b 0xfa 0xbb 0x6c 0x66 0x81 0x89 0x1b 0x59 0xf1 0xa5 0xbf 0xba 0x20 0x5f
25	0x4 0xdd 0x28 0x34 0xf8 0xe0 0xbd 0xdd 0x8 0xd3 0xbd 0x8c 0x2b 0x1c 0x9 0x72
26	0x5d 0x1a 0x94 0x84 0x4f 0x9 0xb9 0x7f 0x5d 0x26 0xd2 0x29 0xc0 0x13 0x6 0xda
27	0x5 0x7c 0x62 0x3c 0xbd 0xfb 0x2b 0x4f 0x77 0x39 0x73 0xb0 0xeb 0x57 0xeb 0x1a
28	0x78 0x8c 0xf8 0x2d 0xd1 0xf 0x9 0x6e 0xf3 0x37 0x27 0x90 0x5a 0xe0 0xce 0xf2
29	0x74 0x12 0xfb 0x68 0x56 0xdd 0x86 0x3e 0xae 0x87 0xb2 0x7a 0xf8 0xea 0x3 0x62

Now we give the table of the distribution of the nonlinearity of the component functions of f_1 generated by the above LFSRs and Keys.

Table 8.3: Distribution of the Nonlinearity of the component functions of f_1

LFSR	Key	Multiplicity	Min
LFSR$_0$	Key_0	{*110(1)109(2)108(15)107(24)106(31)105(35)104(29)103(33)102(29)101(22)100(15)99(6)98(7)97(3)96(2)94(1)92(1)*}	92(1)
LFSR$_0$	Key_1	{*110(1)109(3)108(9)107(22)106(36)105(47)104(41)103(21)102(27)101(15)100(11)99(12)98(4)97(1)96(2)95(2)94(1)91(1)*}	91(1)
LFSR$_0$	Key_2	{*109(7)108(18)107(21)106(24)105(33)104(46)103(31)102(23)101(19)100(11)99(10)98(7)97(2)96(3)95(1)*}	95(1)
LFSR$_0$	Key_3	{*110(1)109(3)108(13)107(16)106(26)105(32)104(49)103(25)102(34)101(27)100(11)99(8)98(5)97(1)96(4)94(1)*}	94(1)
LFSR$_0$	Key_4	{*111(1)110(1)109(2)108(11)107(22)106(35)105(41)104(37)103(35)102(25)101(18)100(14)99(5)98(2)97(2)96(1)95(1)94(1)93(1)92(1)*}	92(1)
LFSR$_0$	Key_5	{*110(4)109(4)108(10)107(22)106(36)105(46)104(34)103(24)102(19)101(20)100(15)99(12)97(7)96(2)95(1)*}	95(1)
LFSR$_0$	Key_6	{*110(1)109(1)108(16)107(20)106(41)105(37)104(33)103(35)102(23)101(21)100(9)99(9)98(3)97(1)96(2)95(1)94(2)92(1)*}	92(1)
LFSR$_0$	Key_7	{*110(1)109(6)108(10)107(21)106(26)105(41)104(33)103(44)102(24)101(15)100(7)99(17)98(4)97(2)96(3)94(2)*}	94(2)
LFSR$_0$	Key_8	{*109(6)108(7)107(22)106(41)105(34)104(44)103(35)102(20)101(17)100(8)99(14)98(2)97(4)96(1)94(1)*}	94(1)
LFSR$_0$	Key_9	{*111(1)109(4)108(8)107(24)106(30)105(37)104(39)103(34)102(30)101(16)100(10)99(6)98(9)97(3)96(2)94(1)93(2)*}	93(2)
LFSR$_0$	Key_{10}	{*110(1)109(2)108(8)107(20)106(34)105(32)104(46)103(39)102(30)101(16)100(12)99(7)97(4)96(1)95(2)93(1)91(1)*}	91(1)
LFSR$_0$	Key_{11}	{*110(1)108(13)107(19)106(33)105(41)104(44)103(37)102(22)101(18)100(10)99(9)98(2)97(6)92(1)*}	92(1)
LFSR$_0$	Key_{12}	{*111(1)109(2)108(12)107(21)106(24)105(36)104(42)103(31)102(28)101(24)100(8)99(11)98(3)97(5)96(3)95(2)94(1)93(1)92(1)*}	92(1)
LFSR$_0$	Key_{13}	{*109(2)108(5)107(25)106(31)105(50)104(41)103(34)102(22)101(16)100(10)99(4)98(7)97(3)96(1)95(3)93(2)*}	93(2)
LFSR$_0$	Key_{14}	{*110(1)109(3)108(10)107(23)106(31)105(49)104(38)103(32)102(22)101(21)100(15)99(5)98(4)97(1)95(1)*}	95(1)
LFSR$_0$	Key_{15}	{*110(1)109(2)108(6)107(23)106(30)105(43)104(44)103(28)102(22)101(22)100(15)99(5)98(6)97(4)96(1)95(3)94(1)*}	94(1)
LFSR$_0$	Key_{16}	{*109(1)108(11)107(24)106(28)105(51)104(36)103(31)102(20)101(27)100(13)99(3)98(4)97(5)95(1)93(1)*}	93(1)
LFSR$_0$	Key_{17}	{*109(4)108(12)107(17)106(37)105(33)104(33)103(33)102(21)101(23)100(12)99(10)98(10)97(3)96(1)95(5)94(1)91(1)*}	91(1)
LFSR$_0$	Key_{18}	{*109(2)108(13)107(21)106(40)105(37)104(40)103(30)102(19)101(20)100(14)99(7)98(6)97(4)95(1)94(1)92(1)*}	92(1)
LFSR$_0$	Key_{19}	{*109(4)108(12)107(18)106(28)105(44)104(36)103(40)102(20)101(23)100(13)99(7)98(1)97(6)96(2)91(1)90(1)*}	90(1)
LFSR$_0$	Key_{20}	{*111(1)109(2)108(14)107(17)106(43)105(41)104(30)103(32)102(24)101(25)100(10)99(5)98(4)97(5)96(1)94(1)91(1)*}	91(1)
LFSR$_0$	Key_{21}	{*109(2)108(5)107(27)106(31)105(43)104(37)103(34)102(24)101(10)100(19)99(10)98(7)97(1)96(2)93(1)91(1)90(1)87(1)*}	87(1)
LFSR$_0$	Key_{22}	{*110(1)109(3)108(5)107(14)106(42)105(37)104(44)103(34)102(25)101(15)100(15)99(8)98(8)96(2)95(2)93(1)*}	93(1)
LFSR$_0$	Key_{23}	{*109(2)108(19)107(21)106(22)105(41)104(47)103(31)102(27)101(16)100(6)99(10)98(4)97(3)96(4)92(1)89(2)*}	89(2)
LFSR$_0$	Key_{24}	{*110(1)109(2)108(6)107(26)106(33)105(42)104(43)103(32)102(24)101(11)100(11)99(9)98(8)97(3)96(3)95(1)94(1)*}	94(1)
LFSR$_0$	Key_{25}	{*110(1)108(10)107(28)106(37)105(38)104(43)103(23)102(21)101(19)100(15)99(7)98(8)97(3)96(1)94(1)90(1)*}	90(1)
LFSR$_0$	Key_{26}	{*110(1)109(1)108(10)107(25)106(22)105(47)104(34)103(35)102(25)101(22)100(11)99(13)98(3)97(5)96(1)95(1)*}	95(1)
LFSR$_0$	Key_{27}	{*110(1)109(3)108(3)107(29)106(28)105(37)104(44)103(36)102(28)101(17)100(10)99(5)98(7)97(5)96(2)93(1)*}	93(1)
LFSR$_0$	Key_{28}	{*110(1)109(4)108(8)107(20)106(41)105(30)104(50)103(28)102(24)101(19)100(12)99(5)98(5)97(2)96(5)94(1)93(1)*}	93(1)
LFSR$_0$	Key_{29}	{*110(2)109(4)108(9)107(17)106(31)105(50)104(34)103(23)102(27)101(22)100(14)99(10)98(4)97(5)96(3)95(1)*}	95(1)
LFSR$_1$	Key_0	{*110(1)109(1)108(10)107(22)106(26)105(46)104(39)103(33)102(17)101(17)100(12)99(12)98(7)97(4)96(2)95(4)94(1)93(2)*}	93(2)
LFSR$_1$	Key_1	{*110(1)109(10)108(10)107(20)106(34)105(41)104(26)103(35)102(32)101(17)100(14)99(7)98(3)97(3)96(1)95(1)94(1)*}	94(1)
LFSR$_1$	Key_2	{*109(4)108(9)107(27)106(47)105(35)104(34)103(24)102(27)101(18)100(13)99(9)98(1)97(4)96(2)94(1)91(1)*}	91(1)
LFSR$_1$	Key_3	{*109(1)108(12)107(23)106(31)105(34)104(46)103(39)102(26)101(19)100(12)99(3)98(4)97(1)96(3)95(1)94(1)*}	94(1)
LFSR$_1$	Key_4	{*109(3)108(5)107(22)106(30)105(34)104(38)103(35)102(31)101(21)100(18)99(7)98(3)97(3)96(3)94(2)85(1)*}	85(1)
LFSR$_1$	Key_5	{*109(3)108(11)107(23)106(34)105(36)104(43)103(27)102(29)101(15)100(17)99(7)98(7)96(2)95(1)94(1)*}	94(1)
LFSR$_1$	Key_6	{*110(1)109(5)108(13)107(24)106(38)105(36)104(28)103(31)102(27)101(18)100(12)99(8)98(8)97(2)96(3)95(1)90(1)*}	90(1)
LFSR$_1$	Key_7	{*109(4)108(12)107(17)106(27)105(41)104(29)103(30)102(44)101(20)100(14)99(5)98(4)97(2)96(4)93(3)*}	93(3)
LFSR$_1$	Key_8	{*110(1)109(2)108(11)107(24)106(29)105(44)104(42)103(27)102(32)101(14)100(9)99(8)98(6)97(3)96(2)95(1)92(1)*}	92(1)
LFSR$_1$	Key_9	{*109(4)108(8)107(15)106(44)105(33)104(42)103(30)102(24)101(21)100(12)99(9)98(4)97(3)96(5)95(1)91(1)*}	91(1)
LFSR$_1$	Key_{10}	{*110(1)109(3)108(11)107(31)106(29)105(39)104(47)103(20)102(25)101(21)100(11)99(5)98(4)97(6)96(2)95(1)*}	95(1)
LFSR$_1$	Key_{11}	{*109(4)108(12)107(21)106(27)105(40)104(37)103(31)102(34)101(16)100(17)99(6)98(3)97(4)96(1)95(1)94(1)90(1)*}	90(1)
LFSR$_1$	Key_{12}	{*109(2)108(9)107(20)106(37)105(39)104(31)103(37)102(23)101(17)100(18)99(5)98(6)97(7)96(3)95(1)94(1)*}	94(1)
LFSR$_1$	Key_{13}	{*110(1)109(2)108(8)107(15)106(41)105(47)104(37)103(37)102(20)101(19)100(12)99(6)98(2)97(3)95(3)94(1)92(1)91(1)*}	91(1)
LFSR$_1$	Key_{14}	{*109(5)108(10)107(23)106(36)105(47)104(34)103(33)102(19)101(21)100(10)99(4)98(9)96(3)95(1)92(1)*}	92(1)
LFSR$_1$	Key_{15}	{*109(1)108(6)107(16)106(31)105(37)104(45)103(32)102(30)101(23)100(16)99(7)98(1)97(5)96(4)95(1)93(1)*}	93(1)
LFSR$_1$	Key_{16}	{*109(1)108(9)107(22)106(36)105(46)104(27)103(36)102(27)101(18)100(14)99(9)98(6)97(2)95(2)92(1)*}	92(1)
LFSR$_1$	Key_{17}	{*109(2)108(15)107(19)106(27)105(42)104(37)103(28)102(36)101(12)100(14)99(10)98(7)97(2)96(1)95(1)94(1)93(1)88(1)*}	88(1)
LFSR$_1$	Key_{18}	{*109(3)108(12)107(18)106(38)105(44)104(38)103(30)102(23)101(20)100(14)99(4)98(7)97(3)93(2)*}	93(2)
LFSR$_1$	Key_{19}	{*109(4)108(6)107(24)106(34)105(35)104(50)103(29)102(23)101(13)100(17)99(5)98(6)97(4)96(4)94(1)92(1)*}	92(1)
LFSR$_1$	Key_{20}	{*109(5)108(8)107(21)106(26)105(47)104(34)103(36)102(23)101(22)100(14)99(13)98(3)97(2)95(1)89(1)*}	89(1)
LFSR$_1$	Key_{21}	{*111(1)109(3)108(8)107(23)106(33)105(38)104(35)103(33)102(25)101(20)100(13)99(10)98(4)97(5)96(1)95(2)94(1)91(1)*}	91(1)
LFSR$_1$	Key_{22}	{*110(1)109(3)108(9)107(15)106(35)105(45)104(40)103(33)102(17)101(20)100(14)99(12)98(3)97(5)96(2)95(2)*}	95(2)
LFSR$_1$	Key_{23}	{*110(1)109(1)108(5)107(19)106(35)105(39)104(37)103(29)102(27)101(22)100(15)99(5)98(11)97(3)96(3)95(2)94(1)92(1)*}	92(1)
LFSR$_1$	Key_{24}	{*109(4)108(5)107(20)106(28)105(45)104(35)103(33)102(26)101(17)100(17)99(9)98(10)97(3)96(2)93(1)91(1)*}	91(1)
LFSR$_1$	Key_{25}	{*109(4)108(11)107(17)106(34)105(33)104(31)103(28)102(30)101(23)100(17)99(11)98(5)97(6)96(3)95(2)89(1)*}	89(1)
LFSR$_1$	Key_{26}	{*110(1)109(5)108(10)107(22)106(38)105(31)104(39)103(38)102(29)101(25)100(8)99(3)98(3)97(2)96(2)*}	96(2)

LFSR_1	Key_{27}	{*109(1)108(12)107(20)106(39)105(44)104(32)103(31)102(27)101(14)100(17)99(10)98(3)97(2)96(1)94(2)93(1)*}	93(1)
LFSR_1	Key_{28}	{*110(1)109(2)108(5)107(22)106(51)105(35)104(43)103(33)102(21)101(14)100(7)99(8)98(6)97(3)96(1)95(3)94(1)*}	94(1)
LFSR_1	Key_{29}	{*109(3)108(7)107(29)106(38)105(33)104(32)103(35)102(32)101(15)100(12)99(8)98(1)97(6)96(1)95(2)94(1)89(1)*}	89(1)
LFSR_2	Key_{0}	{*110(2)109(4)108(7)107(25)106(30)105(36)104(40)103(40)102(25)101(16)100(9)99(8)98(6)97(2)96(3)95(1)93(2)*}	93(2)
LFSR_2	Key_{1}	{*111(1)109(2)108(4)107(22)106(31)105(35)104(39)103(41)102(26)101(17)100(16)99(7)98(8)97(5)94(1)92(1)*}	92(1)
LFSR_2	Key_{2}	{*110(1)109(4)108(5)107(21)106(39)105(43)104(37)103(41)102(26)101(7)100(13)99(6)98(1)97(3)96(4)95(2)94(1)92(1)90(1)*}	90(1)
LFSR_2	Key_{3}	{*109(5)108(13)107(18)106(33)105(40)104(33)103(23)102(30)101(23)100(10)99(14)98(7)97(2)96(2)95(1)94(1)93(1)*}	93(1)
LFSR_2	Key_{4}	{*109(2)108(6)107(22)106(32)105(33)104(30)103(41)102(30)101(16)100(18)99(9)98(7)97(2)96(4)95(3)94(1)*}	94(1)
LFSR_2	Key_{5}	{*109(2)108(9)107(20)106(30)105(30)104(51)103(35)102(26)101(17)100(14)99(6)98(6)97(2)96(2)94(3)93(1)92(2)*}	92(2)
LFSR_2	Key_{6}	{*110(2)109(2)108(8)107(21)106(28)105(38)104(35)103(31)102(29)101(19)100(9)99(14)98(8)97(2)96(1)95(5)94(1)92(2)90(1)*}	90(1)
LFSR_2	Key_{7}	{*109(4)108(6)107(17)106(34)105(51)104(33)103(38)102(24)101(17)100(12)99(10)98(4)97(6)*}	97(6)
LFSR_2	Key_{8}	{*109(5)108(14)107(18)106(30)105(38)104(41)103(35)102(29)101(11)100(8)99(6)98(8)97(7)96(4)95(1)92(1)*}	92(1)
LFSR_2	Key_{9}	{*108(12)107(24)106(34)105(29)104(30)103(31)102(28)101(27)100(19)99(6)98(8)97(4)96(4)*}	96(4)
LFSR_2	Key_{10}	{*109(2)108(9)107(22)106(35)105(43)104(48)103(30)102(18)101(15)100(15)99(7)98(5)97(4)96(2)95(1)*}	95(1)
LFSR_2	Key_{11}	{*109(2)108(12)107(22)106(36)105(43)104(36)103(36)102(25)101(11)100(7)99(13)98(5)97(3)96(2)94(1)93(1)92(1)*}	92(1)
LFSR_2	Key_{12}	{*110(1)109(5)108(14)107(19)106(35)105(37)104(29)103(34)102(19)101(24)100(11)99(13)98(5)97(5)96(3)95(1)93(1)*}	93(1)
LFSR_2	Key_{13}	{*109(2)108(13)107(23)106(26)105(40)104(31)103(36)102(29)101(17)100(16)99(5)98(6)97(7)96(2)95(1)94(1)90(1)*}	90(1)
LFSR_2	Key_{14}	{*111(1)110(1)109(1)108(6)107(23)106(32)105(32)104(39)103(34)102(37)101(13)100(18)99(8)98(5)97(3)96(3)*}	96(3)
LFSR_2	Key_{15}	{*109(2)108(11)107(22)106(27)105(39)104(35)103(38)102(27)101(24)100(14)99(10)98(2)97(2)94(2)92(1)*}	92(1)
LFSR_2	Key_{16}	{*109(4)108(12)107(16)106(28)105(48)104(33)103(42)102(29)101(12)100(10)99(10)98(3)97(2)96(2)95(1)93(2)92(1)91(1)*}	91(1)
LFSR_2	Key_{17}	{*110(1)108(5)107(27)106(34)105(45)104(46)103(32)102(24)101(11)100(11)99(9)98(1)97(2)96(3)95(3)93(2)*}	93(2)
LFSR_2	Key_{18}	{*109(2)108(9)107(18)106(35)105(39)104(34)103(36)102(37)101(15)100(12)99(5)98(6)97(6)96(1)89(1)*}	89(1)
LFSR_2	Key_{19}	{*110(1)109(3)108(5)107(23)106(31)105(34)104(35)103(29)102(20)101(29)100(16)99(13)98(10)97(5)95(1)93(1)*}	93(1)
LFSR_2	Key_{20}	{*109(1)108(17)107(18)106(30)105(32)104(46)103(23)102(35)101(23)100(7)99(10)98(7)97(4)96(1)95(2)*}	95(2)
LFSR_2	Key_{21}	{*109(4)108(6)107(17)106(29)105(45)104(41)103(35)102(27)101(24)100(11)99(8)97(5)96(2)94(1)93(1)*}	93(1)
LFSR_2	Key_{22}	{*110(1)109(1)108(9)107(22)106(33)105(35)104(38)103(35)102(24)101(27)100(10)99(7)98(8)97(2)96(1)95(3)*}	95(3)
LFSR_2	Key_{23}	{*109(4)108(12)107(29)106(37)105(33)104(35)103(30)102(32)101(16)100(8)99(7)98(9)96(2)94(1)92(1)*}	92(1)
LFSR_2	Key_{24}	{*109(4)108(7)107(19)106(32)105(37)104(41)103(40)102(24)101(17)100(16)99(4)98(6)97(4)96(3)94(1)93(1)*}	93(1)
LFSR_2	Key_{25}	{*109(6)108(6)107(18)106(30)105(39)104(38)103(37)102(27)101(14)100(9)99(12)98(7)97(7)96(2)95(1)94(1)93(1)90(1)*}	90(1)
LFSR_2	Key_{26}	{*109(4)108(7)107(19)106(39)105(40)104(42)103(38)102(20)101(14)100(13)99(5)98(2)97(3)96(3)95(2)94(2)93(2)92(1)*}	92(1)
LFSR_2	Key_{27}	{*110(1)109(4)108(6)107(22)106(26)105(51)104(33)103(31)102(22)101(22)100(17)99(7)98(7)97(2)96(1)95(2)93(1)92(1)*}	92(1)
LFSR_2	Key_{28}	{*109(3)108(6)107(28)106(28)105(36)104(45)103(32)102(29)101(21)100(10)99(6)98(5)97(4)95(1)94(2)*}	94(2)
LFSR_2	Key_{29}	{*110(1)109(3)108(10)107(28)106(38)105(34)104(30)103(30)102(27)101(18)100(15)99(10)98(3)97(3)96(2)95(1)94(2)93(1)*}	93(1)
LFSR_3	Key_{0}	{*109(4)108(9)107(21)106(34)105(38)104(33)103(38)102(20)101(17)100(13)99(13)98(6)97(5)95(4)87(1)*}	87(1)
LFSR_3	Key_{1}	{*111(1)109(5)108(8)107(21)106(42)105(33)104(43)103(32)102(20)101(18)100(9)99(6)98(9)97(2)96(3)95(2)93(1)91(1)*}	91(1)
LFSR_3	Key_{2}	{*110(1)109(3)108(12)107(25)106(30)105(33)104(42)103(26)102(20)101(21)100(14)99(8)98(10)97(5)96(4)95(1)93(1)*}	93(1)
LFSR_3	Key_{3}	{*111(1)110(1)109(3)108(11)107(17)106(36)105(41)104(32)103(39)102(29)101(18)100(15)99(6)98(4)97(3)*}	97(3)
LFSR_3	Key_{4}	{*109(1)108(10)107(20)106(36)105(32)104(34)103(37)102(22)101(19)100(16)99(8)98(5)97(6)96(8)95(1)94(1)*}	94(1)
LFSR_3	Key_{5}	{*109(3)108(12)107(24)106(31)105(44)104(34)103(22)102(26)101(24)100(11)99(9)98(5)97(3)96(3)95(1)94(3)93(1)*}	93(1)
LFSR_3	Key_{6}	{*109(2)108(9)107(20)106(25)105(47)104(45)103(25)102(31)101(21)100(15)99(6)98(2)97(2)96(2)95(2)94(1)93(1)*}	93(1)
LFSR_3	Key_{7}	{*110(1)109(4)108(12)107(18)106(24)105(39)104(40)103(34)102(31)101(14)100(9)99(8)98(8)97(7)96(3)95(3)92(1)*}	92(1)
LFSR_3	Key_{8}	{*109(1)108(7)107(17)106(49)105(38)104(41)103(35)102(25)101(13)100(9)99(6)98(6)97(3)96(2)95(1)94(1)93(2)*}	93(2)
LFSR_3	Key_{9}	{*110(1)109(2)108(12)107(19)106(30)105(42)104(41)103(42)102(20)101(18)100(8)99(12)98(6)97(1)95(1)94(1)*}	94(1)
LFSR_3	Key_{10}	{*109(6)108(9)107(24)106(38)105(29)104(32)103(36)102(32)101(15)100(14)99(6)98(6)97(3)96(1)95(2)94(3)*}	94(3)
LFSR_3	Key_{11}	{*109(1)108(5)107(11)106(35)105(40)104(42)103(41)102(28)101(18)100(13)99(7)98(5)97(6)96(1)95(1)94(2)*}	94(2)
LFSR_3	Key_{12}	{*109(2)108(10)107(22)106(31)105(45)104(35)103(33)102(26)101(17)100(14)99(7)98(5)97(1)96(3)95(2)93(2)90(1)*}	90(1)
LFSR_3	Key_{13}	{*110(1)109(1)108(14)107(21)106(35)105(41)104(37)103(35)102(16)101(20)100(7)99(10)98(7)97(8)95(1)94(1)92(1)*}	92(1)
LFSR_3	Key_{14}	{*110(2)109(1)108(12)107(21)106(38)105(37)104(41)103(32)102(19)101(15)100(8)99(10)98(9)97(7)96(2)95(1)92(1)*}	92(1)
LFSR_3	Key_{15}	{*109(5)108(9)107(19)106(23)105(41)104(44)103(33)102(29)101(21)100(6)99(10)98(9)97(3)96(2)95(1)94(1)*}	94(1)
LFSR_3	Key_{16}	{*109(1)108(9)107(22)106(32)105(45)104(34)103(27)102(25)101(22)100(17)99(9)98(4)97(4)96(2)95(2)91(1)*}	91(1)
LFSR_3	Key_{17}	{*110(1)109(1)108(9)107(13)106(37)105(47)104(35)103(40)102(21)101(15)100(14)99(7)98(10)97(2)96(2)95(1)93(1)*}	93(1)
LFSR_3	Key_{18}	{*109(2)108(6)107(25)106(43)105(25)104(42)103(35)102(30)101(15)100(10)99(5)98(7)97(3)96(3)95(4)94(1)*}	94(1)
LFSR_3	Key_{19}	{*109(3)108(8)107(25)106(27)105(51)104(41)103(25)102(29)101(11)100(14)99(6)98(9)97(6)91(1)*}	91(1)
LFSR_3	Key_{20}	{*110(1)109(5)108(5)107(19)106(24)105(45)104(55)103(23)102(24)101(23)100(11)99(3)98(3)97(9)96(1)95(2)94(1)93(2)*}	93(2)
LFSR_3	Key_{21}	{*109(4)108(12)107(27)106(24)105(39)104(40)103(31)102(23)101(19)100(14)99(6)98(6)97(3)96(4)95(3)92(1)*}	92(1)
LFSR_3	Key_{22}	{*109(3)108(8)107(23)106(36)105(40)104(32)103(38)102(22)101(21)100(13)99(8)98(3)97(3)96(4)95(1)93(1)*}	93(1)
LFSR_3	Key_{23}	{*110(1)109(5)108(7)107(22)106(41)105(30)104(39)103(34)102(21)101(19)100(12)99(7)98(8)97(2)96(1)95(3)94(2)93(2)*}	93(2)
LFSR_3	Key_{24}	{*110(1)109(1)108(13)107(22)106(29)105(43)104(32)103(29)102(31)101(23)100(14)99(6)98(6)97(1)96(1)95(3)94(1)*}	94(1)
LFSR_3	Key_{25}	{*109(4)108(15)107(22)106(29)105(48)104(32)103(26)102(26)101(20)100(14)99(7)98(3)97(4)95(1)94(2)93(1)92(1)89(1)*}	89(1)
LFSR_3	Key_{26}	{*110(1)109(5)108(8)107(17)106(31)105(32)104(44)103(30)102(34)101(14)100(16)99(6)98(7)97(6)96(3)95(2)*}	95(2)
LFSR_3	Key_{27}	{*110(1)109(3)108(9)107(27)106(40)105(40)104(42)103(26)102(14)101(19)100(13)99(11)98(4)96(3)95(1)93(1)92(1)90(1)*}	90(1)
LFSR_3	Key_{28}	{*110(1)109(7)108(12)107(18)106(39)105(38)104(36)103(32)102(22)101(10)100(18)99(7)98(5)97(7)96(1)95(1)93(1)92(1)*}	92(1)
LFSR_3	Key_{29}	{*111(1)109(3)108(11)107(17)106(37)105(38)104(36)103(30)102(28)101(14)100(21)99(7)98(4)97(2)96(3)95(2)93(1)92(1)*}	92(1)

LFSR_4	Key_0	{*110(2)109(7)108(9)107(23)106(32)105(39)104(44)103(22)102(25)101(18)100(8)99(15)98(4)97(3)96(3)95(1)91(1)*}	91(1)
LFSR_4	Key_1	{*110(2)109(4)108(14)107(20)106(38)105(29)104(42)103(30)102(23)101(19)100(16)99(4)98(7)97(4)96(1)95(2)92(1)*}	92(1)
LFSR_4	Key_2	{*109(5)108(11)107(13)106(31)105(27)104(42)103(27)102(26)101(26)100(16)99(11)98(10)97(6)96(2)95(1)94(1)89(1)*}	89(1)
LFSR_4	Key_3	{*109(1)108(14)107(15)106(35)105(40)104(33)103(35)102(27)101(19)100(13)99(5)98(8)97(5)96(4)94(2)*}	94(2)
LFSR_4	Key_4	{*110(1)109(4)108(10)107(20)106(35)105(33)104(40)103(30)102(34)101(21)100(13)99(4)98(4)97(2)96(1)95(3)88(1)*}	88(1)
LFSR_4	Key_5	{*109(5)108(6)107(23)106(44)105(28)104(34)103(24)102(33)101(18)100(17)99(8)98(2)97(5)96(4)95(4)93(1)*}	93(1)
LFSR_4	Key_6	{*110(1)109(2)108(7)107(19)106(26)105(34)104(41)103(35)102(27)101(22)100(14)99(11)98(8)97(4)96(1)95(2)93(1)92(1)*}	92(1)
LFSR_4	Key_7	{*110(2)109(2)108(4)107(19)106(38)105(47)104(38)103(36)102(30)101(12)100(11)99(4)98(4)97(3)96(1)95(2)94(2)91(1)*}	91(1)
LFSR_4	Key_8	{*109(5)108(7)107(25)106(34)105(44)104(35)103(37)102(28)101(16)100(14)99(3)98(4)97(2)96(1)95(1)*}	95(1)
LFSR_4	Key_9	{*111(1)109(3)108(9)107(15)106(29)105(36)104(41)103(38)102(18)101(21)100(16)99(9)98(7)97(4)95(4)94(3)93(2)*}	93(2)
LFSR_4	Key_{10}	{*109(4)108(12)107(24)106(29)105(41)104(29)103(38)102(25)101(16)100(13)99(8)98(3)97(9)96(2)95(2)93(1)*}	93(1)
LFSR_4	Key_{11}	{*110(1)109(4)108(9)107(24)106(38)105(38)104(32)103(20)102(36)101(15)100(10)99(9)98(6)97(4)95(5)94(3)93(1)90(1)*}	90(1)
LFSR_4	Key_{12}	{*109(4)108(11)107(24)106(32)105(43)104(26)103(29)102(25)101(23)100(19)99(8)98(4)97(2)96(4)95(1)93(1)*}	93(1)
LFSR_4	Key_{13}	{*109(2)108(11)107(23)106(37)105(38)104(30)103(34)102(28)101(20)100(12)99(6)98(5)97(1)96(4)95(2)94(1)93(1)91(1)*}	91(1)
LFSR_4	Key_{14}	{*109(5)108(12)107(20)106(35)105(36)104(35)103(39)102(21)101(21)100(12)99(9)98(5)97(3)96(2)94(1)*}	94(1)
LFSR_4	Key_{15}	{*109(3)108(11)107(17)106(28)105(46)104(35)103(31)102(32)101(16)100(13)99(12)98(8)97(3)96(1)*}	96(1)
LFSR_4	Key_{16}	{*108(9)107(21)106(38)105(39)104(32)103(23)102(36)101(17)100(19)99(4)98(8)97(3)96(5)94(1)93(1)*}	93(1)
LFSR_4	Key_{17}	{*109(1)108(5)107(24)106(28)105(38)104(31)103(37)102(30)101(27)100(11)99(9)98(4)97(5)96(3)94(1)90(1)84(1)*}	84(1)
LFSR_4	Key_{18}	{*110(1)109(2)108(11)107(23)106(31)105(39)104(40)103(23)102(23)101(24)100(13)99(11)98(7)97(5)96(2)95(1)*}	95(1)
LFSR_4	Key_{19}	{*109(3)108(13)107(21)106(33)105(30)104(35)103(30)102(30)101(22)100(12)99(10)98(3)97(4)96(3)95(2)94(2)93(2)91(1)*}	91(1)
LFSR_4	Key_{20}	{*109(4)108(14)107(26)106(28)105(25)104(39)103(26)102(23)101(27)100(17)99(12)98(8)97(3)96(4)*}	96(4)
LFSR_4	Key_{21}	{*109(3)108(6)107(12)106(20)105(42)104(43)103(32)102(30)101(25)100(15)99(10)98(9)97(3)96(1)95(3)91(1)90(1)*}	90(1)
LFSR_4	Key_{22}	{*110(3)109(8)108(12)107(13)106(34)105(37)104(39)103(36)102(24)101(19)100(11)99(3)98(8)97(2)96(5)95(1)94(1)*}	94(1)
LFSR_4	Key_{23}	{*109(2)108(7)107(24)106(31)105(35)104(43)103(27)102(26)101(17)100(18)99(7)98(6)97(3)96(4)95(2)92(4)*}	92(4)
LFSR_4	Key_{24}	{*109(4)108(6)107(20)106(35)105(37)104(45)103(36)102(24)101(14)100(6)99(13)98(5)97(4)96(2)95(2)94(1)92(1)90(1)*}	90(1)
LFSR_4	Key_{25}	{*109(1)108(13)107(14)106(36)105(39)104(31)103(27)102(36)101(27)100(12)99(4)98(3)97(4)96(6)95(1)94(1)93(1)*}	93(1)
LFSR_4	Key_{26}	{*109(3)108(6)107(24)106(32)105(38)104(28)103(48)102(23)101(17)100(14)99(4)98(6)97(3)96(6)95(2)94(1)93(1)*}	93(1)
LFSR_4	Key_{27}	{*109(2)108(7)107(19)106(36)105(36)104(45)103(29)102(29)101(16)100(10)99(6)98(10)97(4)96(4)95(1)94(1)92(1)*}	92(1)
LFSR_4	Key_{28}	{*109(3)108(16)107(13)106(27)105(39)104(42)103(41)102(31)101(17)100(7)99(10)98(3)97(2)96(2)95(2)92(1)*}	92(1)
LFSR_4	Key_{29}	{*110(1)108(11)107(21)106(26)105(49)104(40)103(32)102(21)101(15)100(20)99(3)98(9)97(4)96(2)95(2)*}	95(2)
LFSR_5	Key_0	{*109(4)108(10)107(20)106(28)105(39)104(48)103(27)102(32)101(15)100(12)99(6)98(6)97(1)96(6)95(1)93(1)*}	93(1)
LFSR_5	Key_1	{*110(2)109(4)108(14)107(28)106(32)105(41)104(35)103(24)102(26)101(25)100(9)99(7)98(4)96(2)95(1)94(1)92(1)*}	92(1)
LFSR_5	Key_2	{*110(1)109(3)108(9)107(22)106(41)105(36)104(27)103(31)102(33)101(18)100(11)99(7)98(8)97(4)96(1)95(1)94(2)90(1)*}	90(1)
LFSR_5	Key_3	{*108(11)107(24)106(33)105(48)104(36)103(35)102(26)101(12)100(12)99(4)98(3)97(6)96(1)95(2)94(2)93(1)*}	93(1)
LFSR_5	Key_4	{*110(1)109(4)108(7)107(25)106(22)105(40)104(39)103(40)102(24)101(16)100(12)99(7)98(6)97(9)96(3)94(1)*}	94(1)
LFSR_5	Key_5	{*109(2)108(8)107(27)106(39)105(36)104(37)103(31)102(20)101(21)100(15)99(7)98(2)97(6)96(1)95(2)94(1)90(1)*}	90(1)
LFSR_5	Key_6	{*109(3)108(10)107(20)106(40)105(44)104(23)103(37)102(21)101(21)100(16)99(10)98(2)97(2)96(3)95(2)94(1)89(1)*}	89(1)
LFSR_5	Key_7	{*109(1)108(12)107(21)106(42)105(41)104(38)103(37)102(21)101(15)100(8)99(7)98(3)97(1)96(5)95(2)92(2)*}	92(2)
LFSR_5	Key_8	{*110(1)109(3)108(11)107(16)106(39)105(35)104(40)103(28)102(35)101(16)100(9)99(9)98(7)97(4)96(1)95(1)94(1)*}	94(1)
LFSR_5	Key_9	{*110(2)109(3)108(5)107(23)106(33)105(48)104(26)103(31)102(33)101(21)100(12)99(10)98(7)97(2)*}	97(2)
LFSR_5	Key_{10}	{*109(3)108(7)107(15)106(33)105(53)104(37)103(30)102(29)101(18)100(15)99(8)98(3)97(2)93(2)91(1)*}	91(1)
LFSR_5	Key_{11}	{*109(1)108(16)107(20)106(33)105(34)104(41)103(33)102(38)101(15)100(8)99(9)98(1)97(2)96(3)94(1)92(1)*}	92(1)
LFSR_5	Key_{12}	{*108(14)107(24)106(39)105(38)104(38)103(30)102(18)101(17)100(14)99(12)98(6)97(3)96(1)95(1)91(1)*}	91(1)
LFSR_5	Key_{13}	{*110(1)109(1)108(18)107(26)106(31)105(38)104(49)103(24)102(20)101(16)100(9)99(5)98(5)97(6)96(4)95(2)93(1)*}	93(1)
LFSR_5	Key_{14}	{*109(2)108(7)107(18)106(43)105(31)104(42)103(34)102(30)101(22)100(11)99(9)98(3)97(1)96(2)91(1)*}	91(1)
LFSR_5	Key_{15}	{*109(3)108(9)107(21)106(31)105(30)104(41)103(36)102(34)101(15)100(13)99(7)98(3)97(8)96(1)95(2)94(2)*}	94(2)
LFSR_5	Key_{16}	{*108(11)107(22)106(30)105(35)104(30)103(47)102(26)101(16)100(17)99(9)98(5)97(5)96(2)94(1)*}	94(1)
LFSR_5	Key_{17}	{*110(1)109(2)108(9)107(17)106(30)105(40)104(43)103(38)102(25)101(26)100(12)99(6)98(2)97(3)96(1)94(1)*}	94(1)
LFSR_5	Key_{18}	{*110(1)109(4)108(8)107(16)106(34)105(29)104(45)103(46)102(23)101(18)100(12)99(7)98(4)97(6)96(1)91(1)90(1)*}	90(1)
LFSR_5	Key_{19}	{*109(2)108(10)107(18)106(36)105(51)104(35)103(30)102(22)101(20)100(10)99(9)98(7)97(3)95(1)94(2)*}	94(2)
LFSR_5	Key_{20}	{*109(4)108(12)107(33)106(33)105(31)104(25)103(36)102(21)101(18)100(12)99(11)98(11)97(3)96(4)95(1)92(1)*}	92(1)
LFSR_5	Key_{21}	{*110(1)109(3)108(11)107(21)106(39)105(30)104(37)103(34)102(22)101(18)100(17)99(9)98(7)97(4)95(1)93(2)*}	93(2)
LFSR_5	Key_{22}	{*111(1)109(2)108(13)107(22)106(35)105(46)104(31)103(32)102(20)101(18)100(13)99(14)98(4)97(1)96(2)95(1)93(1)*}	93(1)
LFSR_5	Key_{23}	{*110(1)109(3)108(6)107(20)106(33)105(37)104(40)103(34)102(30)101(21)100(12)99(10)98(6)97(3)*}	97(3)
LFSR_5	Key_{24}	{*109(3)108(11)107(24)106(23)105(40)104(32)103(36)102(24)101(25)100(12)99(12)98(4)97(4)96(2)95(2)94(1)93(1)*}	93(1)
LFSR_5	Key_{25}	{*109(3)108(9)107(18)106(31)105(36)104(36)103(23)102(23)101(28)100(18)99(11)98(7)97(5)96(3)95(3)92(1)90(1)*}	90(1)
LFSR_5	Key_{26}	{*110(1)109(3)108(8)107(25)106(21)105(36)104(36)103(35)102(30)101(27)100(13)99(5)98(4)97(8)96(1)94(1)93(1)86(1)*}	86(1)
LFSR_5	Key_{27}	{*110(2)108(8)107(17)106(39)105(49)104(31)103(33)102(24)101(13)100(16)99(9)98(6)97(4)96(2)95(1)94(1)93(1)*}	93(1)
LFSR_5	Key_{28}	{*109(2)108(9)107(25)106(22)105(38)104(40)103(32)102(23)101(26)100(10)99(7)98(9)97(5)96(4)95(3)88(1)*}	88(1)
LFSR_5	Key_{29}	{*109(4)108(10)107(22)106(41)105(35)104(38)103(24)102(28)101(24)100(9)99(5)98(3)97(10)95(3)*}	95(3)
LFSR_6	Key_0	{*109(3)108(11)107(21)106(36)105(39)104(40)103(26)102(23)101(19)100(13)99(10)98(2)97(3)96(1)95(3)94(4)92(1)90(1)*}	90(1)
LFSR_6	Key_1	{*110(1)109(2)108(9)107(21)106(32)105(43)104(32)103(35)102(24)101(17)100(17)99(10)98(7)97(2)96(2)95(1)94(1)*}	94(1)
LFSR_6	Key_2	{*109(3)108(18)107(23)106(28)105(37)104(42)103(28)102(25)101(17)100(13)99(7)98(5)97(2)96(2)95(4)94(1)89(1)*}	89(1)

LFSR$_6$	Key_3	{∗109(1)108(10)107(24)106(30)105(40)104(39)103(35)102(28)101(17)100(9)99(8)98(6)97(4)96(3)95(1)93(1)∗}	93(1)
LFSR$_6$	Key_4	{∗109(6)108(14)107(24)106(35)105(23)104(34)103(32)102(26)101(20)100(15)99(12)98(7)97(4)96(2)95(1)92(1)∗}	92(1)
LFSR$_6$	Key_5	{∗109(2)108(4)107(18)106(43)105(40)104(37)103(35)102(22)101(17)100(8)99(10)98(4)97(7)96(5)95(2)93(1)86(1)∗}	86(1)
LFSR$_6$	Key_6	{∗110(1)109(1)108(9)107(21)106(29)105(39)104(47)103(39)102(26)101(12)100(13)99(8)98(2)97(4)96(3)95(1)92(1)∗}	92(1)
LFSR$_6$	Key_7	{∗111(1)109(4)108(8)107(23)106(28)105(38)104(43)103(42)102(16)101(22)100(14)99(8)98(7)97(1)95(1)∗}	95(1)
LFSR$_6$	Key_8	{∗109(4)108(6)107(20)106(34)105(42)104(31)103(33)102(18)101(33)100(14)99(6)98(3)97(6)96(3)95(1)92(1)91(1)∗}	91(1)
LFSR$_6$	Key_9	{∗109(1)108(8)107(26)106(37)105(37)104(34)103(28)102(26)101(21)100(9)99(13)98(6)97(5)96(3)95(1)94(1)∗}	94(1)
LFSR$_6$	Key_{10}	{∗110(2)109(6)108(9)107(17)106(28)105(46)104(35)103(29)102(22)101(21)100(15)99(16)98(4)97(3)96(2)92(1)∗}	92(1)
LFSR$_6$	Key_{11}	{∗109(4)108(7)107(25)106(34)105(35)104(38)103(37)102(31)101(19)100(8)99(6)98(6)97(3)95(2)93(1)∗}	93(1)
LFSR$_6$	Key_{12}	{∗109(2)108(11)107(15)106(33)105(40)104(38)103(37)102(21)101(24)100(17)99(6)98(3)96(3)95(2)94(2)93(1)86(1)∗}	86(1)
LFSR$_6$	Key_{13}	{∗109(1)108(9)107(24)106(29)105(44)104(38)103(38)102(17)101(21)100(11)99(10)98(2)97(1)96(3)95(4)94(4)∗}	94(4)
LFSR$_6$	Key_{14}	{∗110(1)109(3)108(12)107(22)106(31)105(42)104(41)103(37)102(16)101(14)100(14)99(10)98(7)97(4)96(1)88(1)∗}	88(1)
LFSR$_6$	Key_{15}	{∗109(3)108(14)107(23)106(33)105(42)104(34)103(37)102(21)101(13)100(13)99(10)98(7)97(2)96(1)95(2)94(1)∗}	94(1)
LFSR$_6$	Key_{16}	{∗109(4)108(12)107(24)106(31)105(34)104(37)103(45)102(31)101(9)100(13)99(5)98(6)97(3)94(1)91(1)∗}	91(1)
LFSR$_6$	Key_{17}	{∗109(3)108(13)107(24)106(34)105(25)104(49)103(20)102(37)101(24)100(12)99(5)98(2)97(4)96(3)94(1)∗}	94(1)
LFSR$_6$	Key_{18}	{∗109(4)108(9)107(23)106(24)105(44)104(49)103(35)102(20)101(14)100(17)99(2)98(5)97(2)96(3)95(1)94(3)93(1)∗}	93(1)
LFSR$_6$	Key_{19}	{∗109(6)108(9)107(25)106(31)105(32)104(39)103(33)102(30)101(17)100(10)99(8)98(5)97(5)96(5)94(1)∗}	94(1)
LFSR$_6$	Key_{20}	{∗109(1)108(7)107(11)106(35)105(33)104(51)103(35)102(27)101(19)100(11)99(8)98(3)97(8)96(2)95(1)94(2)93(1)91(1)∗}	91(1)
LFSR$_6$	Key_{21}	{∗109(3)108(19)107(16)106(25)105(40)104(35)103(44)102(28)101(22)100(12)99(4)98(4)97(2)95(2)∗}	95(2)
LFSR$_6$	Key_{22}	{∗109(1)108(14)107(21)106(36)105(37)104(32)103(30)102(28)101(16)100(12)99(12)98(7)97(5)96(1)95(4)∗}	95(4)
LFSR$_6$	Key_{23}	{∗110(1)109(4)108(10)107(27)106(43)105(33)104(35)103(32)102(25)101(21)100(11)99(5)98(5)97(1)96(1)95(2)∗}	95(2)
LFSR$_6$	Key_{24}	{∗110(1)109(3)108(7)107(18)106(34)105(44)104(31)103(25)102(27)101(23)100(14)99(12)98(8)97(5)96(2)94(1)90(1)∗}	90(1)
LFSR$_6$	Key_{25}	{∗110(2)109(4)108(12)107(15)106(31)105(37)104(38)103(37)102(22)101(19)100(10)99(10)98(13)97(2)96(2)95(2)∗}	95(2)
LFSR$_6$	Key_{26}	{∗108(15)107(17)106(35)105(37)104(38)103(34)102(32)101(13)100(17)99(9)98(6)97(1)96(2)∗}	96(2)
LFSR$_6$	Key_{27}	{∗111(1)109(6)108(10)107(14)106(36)105(40)104(46)103(32)102(18)101(25)100(11)99(6)98(5)97(3)96(2)90(1)∗}	90(1)
LFSR$_6$	Key_{28}	{∗111(1)109(4)108(9)107(18)106(36)105(37)104(35)103(35)102(19)101(15)100(9)99(13)98(12)97(6)96(2)95(2)94(1)93(1)90(1)∗}	90(1)
LFSR$_6$	Key_{29}	{∗110(3)109(3)108(13)107(22)106(35)105(31)104(42)103(29)102(29)101(19)100(15)99(1)98(4)97(4)96(2)95(2)94(1)93(1)∗}	93(1)
LFSR$_7$	Key_0	{∗108(7)107(20)106(33)105(44)104(42)103(39)102(22)101(20)100(10)99(7)98(2)97(5)95(2)93(3)∗}	93(3)
LFSR$_7$	Key_1	{∗109(2)108(11)107(22)106(38)105(39)104(37)103(33)102(27)101(20)100(10)99(4)98(6)97(2)96(2)95(3)∗}	95(3)
LFSR$_7$	Key_2	{∗110(1)109(5)108(10)107(27)106(28)105(52)104(36)103(25)102(29)101(13)100(12)99(9)98(4)96(3)95(1)94(1)∗}	94(1)
LFSR$_7$	Key_3	{∗110(1)109(1)108(9)107(21)106(32)105(34)104(33)103(34)102(26)101(28)100(11)99(8)98(7)97(6)96(1)95(3)94(1)∗}	94(1)
LFSR$_7$	Key_4	{∗110(1)109(3)108(9)107(14)106(32)105(38)104(43)103(34)102(29)101(16)100(7)99(14)98(5)97(5)96(4)94(2)∗}	94(2)
LFSR$_7$	Key_5	{∗109(3)108(9)107(20)106(30)105(35)104(42)103(36)102(26)101(14)100(13)99(12)98(4)97(3)96(4)95(3)93(1)92(1)∗}	92(1)
LFSR$_7$	Key_6	{∗109(3)108(10)107(21)106(26)105(38)104(48)103(40)102(26)101(19)100(8)99(6)98(4)97(2)96(3)94(1)93(1)∗}	93(1)
LFSR$_7$	Key_7	{∗110(1)109(7)108(11)107(17)106(34)105(45)104(31)103(39)102(15)101(15)100(13)99(14)98(3)97(7)96(2)95(2)∗}	95(2)
LFSR$_7$	Key_8	{∗110(1)109(6)108(18)107(26)106(24)105(37)104(38)103(46)102(20)101(16)100(10)99(7)98(3)97(1)96(1)94(1)93(1)∗}	93(1)
LFSR$_7$	Key_9	{∗109(4)108(7)107(26)106(26)105(31)104(35)103(44)102(26)101(20)100(19)99(4)98(6)97(4)96(2)95(2)∗}	95(2)
LFSR$_7$	Key_{10}	{∗109(3)108(6)107(11)106(33)105(42)104(32)103(31)102(31)101(23)100(16)99(12)98(8)97(3)96(3)95(1)94(1)∗}	94(1)
LFSR$_7$	Key_{11}	{∗110(3)108(9)107(27)106(32)105(36)104(47)103(33)102(17)101(17)100(13)99(10)98(5)97(5)94(2)∗}	94(2)
LFSR$_7$	Key_{12}	{∗109(6)108(9)107(28)106(30)105(41)104(40)103(24)102(19)101(33)100(7)99(6)98(4)97(5)96(1)95(1)92(2)∗}	92(2)
LFSR$_7$	Key_{13}	{∗109(2)108(7)107(25)106(30)105(43)104(39)103(28)102(18)101(30)100(8)99(11)98(3)97(7)96(2)95(1)93(2)∗}	93(2)
LFSR$_7$	Key_{14}	{∗109(3)108(9)107(28)106(31)105(35)104(30)103(36)102(29)101(24)100(8)99(13)98(8)96(1)95(1)∗}	95(1)
LFSR$_7$	Key_{15}	{∗109(4)108(7)107(25)106(24)105(40)104(36)103(39)102(33)101(17)100(10)99(7)98(7)97(2)96(1)95(3)90(1)∗}	90(1)
LFSR$_7$	Key_{16}	{∗109(5)108(10)107(20)106(35)105(40)104(30)103(38)102(29)101(17)100(10)99(6)98(6)97(5)96(3)95(1)93(1)∗}	93(1)
LFSR$_7$	Key_{17}	{∗110(1)109(3)108(10)107(32)106(35)105(28)104(37)103(24)102(24)101(19)100(17)99(15)98(5)97(1)96(1)95(2)94(1)92(1)∗}	92(1)
LFSR$_7$	Key_{18}	{∗109(3)108(11)107(20)106(25)105(31)104(35)103(44)102(31)101(20)100(15)99(6)98(5)97(6)96(3)94(1)∗}	94(1)
LFSR$_7$	Key_{19}	{∗109(3)108(11)107(17)106(39)105(44)104(39)103(34)102(25)101(18)100(7)99(8)98(2)97(5)94(2)93(1)91(1)∗}	91(1)
LFSR$_7$	Key_{20}	{∗109(2)108(13)107(18)106(38)105(43)104(46)103(28)102(25)101(16)100(15)99(4)98(3)97(3)96(1)95(1)∗}	95(1)
LFSR$_7$	Key_{21}	{∗109(2)108(7)107(16)106(38)105(50)104(34)103(30)102(31)101(19)100(11)99(8)98(1)97(2)96(2)95(2)94(1)93(1)92(1)∗}	92(1)
LFSR$_7$	Key_{22}	{∗110(1)109(4)108(6)107(10)106(31)105(46)104(41)103(27)102(36)101(26)100(8)99(8)98(6)97(2)96(1)94(1)93(1)92(1)∗}	92(1)
LFSR$_7$	Key_{23}	{∗108(10)107(18)106(42)105(46)104(37)103(44)102(18)101(13)100(9)99(9)98(4)97(3)96(1)95(1)89(1)∗}	89(1)
LFSR$_7$	Key_{24}	{∗110(1)109(3)108(17)107(13)106(35)105(37)104(35)103(34)102(31)101(18)100(15)99(6)98(4)97(3)96(3)95(1)∗}	95(1)
LFSR$_7$	Key_{25}	{∗110(1)109(3)108(14)107(18)106(37)105(37)104(34)103(46)102(21)101(18)100(12)99(6)98(5)97(3)96(1)∗}	96(1)
LFSR$_7$	Key_{26}	{∗109(1)108(7)107(22)106(36)105(39)104(38)103(38)102(39)101(11)100(4)99(5)98(7)97(3)96(1)95(3)94(1)93(1)∗}	93(1)
LFSR$_7$	Key_{27}	{∗109(1)108(12)107(20)106(33)105(37)104(30)103(31)102(31)101(22)100(14)99(11)98(7)97(2)96(2)95(1)94(2)∗}	94(2)
LFSR$_7$	Key_{28}	{∗109(2)108(5)107(13)106(38)105(38)104(44)103(40)102(32)101(19)100(8)99(7)98(2)97(3)96(1)95(3)92(1)∗}	92(1)
LFSR$_7$	Key_{29}	{∗108(8)107(18)106(38)105(41)104(41)103(33)102(19)101(24)100(15)99(6)98(8)97(2)95(1)94(1)93(1)∗}	93(1)
LFSR$_8$	Key_0	{∗110(1)108(8)107(18)106(39)105(47)104(37)103(28)102(20)101(16)100(15)99(15)98(7)97(2)96(1)94(1)91(1)∗}	91(1)
LFSR$_8$	Key_1	{∗111(1)109(4)108(12)107(17)106(40)105(33)104(41)103(37)102(23)101(14)100(16)99(9)98(2)97(5)96(1)91(1)∗}	91(1)
LFSR$_8$	Key_2	{∗108(16)107(17)106(33)105(36)104(37)103(22)102(36)101(24)100(7)99(11)98(6)97(6)96(2)95(1)92(2)∗}	92(2)
LFSR$_8$	Key_3	{∗109(3)108(11)107(22)106(21)105(42)104(44)103(35)102(28)101(25)100(7)99(8)98(4)97(2)96(2)95(1)92(1)∗}	92(1)
LFSR$_8$	Key_4	{∗109(3)108(14)107(23)106(31)105(44)104(46)103(25)102(23)101(17)100(15)99(6)98(4)97(2)96(1)95(1)92(1)∗}	92(1)
LFSR$_8$	Key_5	{∗108(15)107(26)106(30)105(33)104(43)103(37)102(27)101(14)100(9)99(10)98(3)97(3)96(2)95(1)93(2)90(1)∗}	90(1)

LFSR$_8$	Key_6	{∗108(11)107(24)106(33)105(52)104(33)103(31)102(17)101(13)100(17)99(9)98(5)97(4)95(1)94(3)93(2)92(1)∗}	92(1)
LFSR$_8$	Key_7	{∗109(4)108(10)107(21)106(34)105(42)104(38)103(35)102(22)101(18)100(11)99(6)98(6)97(3)96(3)95(3)∗}	95(3)
LFSR$_8$	Key_8	{∗109(3)108(13)107(29)106(31)105(46)104(34)103(22)102(28)101(20)100(10)99(6)98(7)97(4)96(1)94(1)93(1)∗}	93(1)
LFSR$_8$	Key_9	{∗109(2)108(7)107(21)106(25)105(43)104(37)103(31)102(20)101(27)100(14)99(12)98(7)97(4)96(4)95(1)94(1)∗}	94(1)
LFSR$_8$	Key_{10}	{∗109(3)108(8)107(31)106(33)105(37)104(35)103(29)102(20)101(19)100(20)99(7)98(7)97(2)96(3)95(1)92(1)∗}	92(1)
LFSR$_8$	Key_{11}	{∗109(2)108(10)107(23)106(26)105(43)104(37)103(36)102(34)101(18)100(13)99(4)98(3)97(2)96(2)95(1)92(1)90(1)∗}	90(1)
LFSR$_8$	Key_{12}	{∗109(2)108(11)107(21)106(37)105(29)104(40)103(39)102(22)101(21)100(12)99(10)98(3)97(3)96(1)95(2)94(3)∗}	94(3)
LFSR$_8$	Key_{13}	{∗110(2)108(12)107(21)106(27)105(37)104(33)103(30)102(30)101(27)100(18)99(5)98(5)97(4)96(1)95(3)94(1)∗}	94(1)
LFSR$_8$	Key_{14}	{∗109(2)108(9)107(24)106(33)105(49)104(29)103(33)102(20)101(20)100(13)99(10)98(5)97(6)96(1)95(2)∗}	95(2)
LFSR$_8$	Key_{15}	{∗110(1)109(1)108(6)107(16)106(39)105(37)104(44)103(36)102(27)101(15)100(17)99(5)98(3)97(6)96(1)95(2)∗}	95(2)
LFSR$_8$	Key_{16}	{∗110(1)109(4)108(9)107(20)106(36)105(37)104(44)103(33)102(22)101(22)100(8)99(7)98(7)96(3)95(1)92(2)∗}	92(2)
LFSR$_8$	Key_{17}	{∗110(1)109(3)108(7)107(20)106(41)105(40)104(39)103(25)102(32)101(11)100(13)99(10)98(6)97(3)96(3)94(1)93(1)∗}	93(1)
LFSR$_8$	Key_{18}	{∗109(2)108(13)107(23)106(24)105(42)104(30)103(35)102(31)101(16)100(17)99(7)98(7)97(1)96(5)95(2)93(1)∗}	93(1)
LFSR$_8$	Key_{19}	{∗109(8)108(6)107(28)106(22)105(40)104(36)103(38)102(22)101(21)100(10)99(10)98(4)97(4)96(4)95(2)93(1)∗}	93(1)
LFSR$_8$	Key_{20}	{∗109(2)108(9)107(15)106(32)105(44)104(33)103(37)102(27)101(19)100(8)99(13)98(7)97(2)96(2)95(2)94(1)93(2)92(1)∗}	92(1)
LFSR$_8$	Key_{21}	{∗109(6)108(8)107(20)106(30)105(49)104(33)103(36)102(30)101(19)100(10)99(8)98(3)97(4)∗}	97(4)
LFSR$_8$	Key_{22}	{∗109(2)108(15)107(23)106(34)105(43)104(25)103(29)102(25)101(17)100(9)99(7)98(7)97(9)96(4)95(5)94(2)∗}	94(2)
LFSR$_8$	Key_{23}	{∗110(2)109(3)108(5)107(27)106(33)105(43)104(23)103(35)102(32)101(13)100(12)99(10)98(9)97(5)96(1)94(1)93(1)89(1)∗}	89(1)
LFSR$_8$	Key_{24}	{∗110(1)109(5)108(5)107(20)106(40)105(40)104(38)103(34)102(30)101(13)100(10)99(6)98(3)97(4)96(4)95(1)94(1)93(1)∗}	93(1)
LFSR$_8$	Key_{25}	{∗110(1)109(3)108(10)107(26)106(37)105(39)104(38)103(21)102(30)101(15)100(17)99(9)98(7)97(1)94(1)89(1)∗}	89(1)
LFSR$_8$	Key_{26}	{∗109(3)108(11)107(28)106(32)105(34)104(40)103(34)102(27)101(12)100(10)99(13)98(7)97(2)96(1)95(2)∗}	95(2)
LFSR$_8$	Key_{27}	{∗109(3)108(7)107(24)106(33)105(47)104(31)103(35)102(20)101(17)100(20)99(9)98(4)97(5)95(1)∗}	95(1)
LFSR$_8$	Key_{28}	{∗109(1)108(12)107(15)106(36)105(32)104(41)103(36)102(24)101(22)100(19)99(8)98(3)97(5)95(1)89(1)∗}	89(1)
LFSR$_8$	Key_{29}	{∗109(1)108(15)107(18)106(35)105(27)104(44)103(28)102(28)101(17)100(22)99(8)98(3)97(4)96(4)93(1)92(1)∗}	92(1)
LFSR$_9$	Key_0	{∗110(1)109(2)108(5)107(22)106(32)105(48)104(48)103(28)102(22)101(14)100(14)99(5)98(5)97(3)96(2)95(1)94(2)92(2)∗}	92(2)
LFSR$_9$	Key_1	{∗110(1)109(3)108(9)107(20)106(29)105(44)104(41)103(33)102(26)101(23)100(8)99(9)98(7)97(1)96(2)∗}	96(2)
LFSR$_9$	Key_2	{∗110(1)109(3)108(8)107(17)106(34)105(42)104(36)103(46)102(24)101(19)100(7)99(9)98(5)97(2)95(3)∗}	95(3)
LFSR$_9$	Key_3	{∗110(1)109(4)108(10)107(18)106(34)105(43)104(29)103(32)102(35)101(17)100(14)99(8)98(4)97(3)96(2)94(1)93(1)∗}	93(1)
LFSR$_9$	Key_4	{∗109(4)108(5)107(22)106(39)105(28)104(32)103(37)102(30)101(27)100(14)99(9)98(4)97(1)96(2)95(2)∗}	95(2)
LFSR$_9$	Key_5	{∗108(13)107(32)106(37)105(34)104(32)103(32)102(26)101(23)100(13)99(3)98(2)97(3)96(1)95(3)92(2)∗}	92(2)
LFSR$_9$	Key_6	{∗109(2)108(10)107(19)106(34)105(43)104(42)103(28)102(27)101(14)100(13)99(12)98(6)97(3)96(2)95(1)∗}	95(1)
LFSR$_9$	Key_7	{∗110(1)109(3)108(8)107(24)106(37)105(38)104(39)103(35)102(23)101(19)100(7)99(4)98(4)97(7)96(2)95(3)94(1)93(1)∗}	93(1)
LFSR$_9$	Key_8	{∗109(3)108(11)107(16)106(33)105(33)104(37)103(38)102(26)101(27)100(9)99(14)98(5)97(3)96(1)∗}	96(1)
LFSR$_9$	Key_9	{∗108(5)107(22)106(27)105(42)104(49)103(37)102(26)101(18)100(8)99(9)98(8)97(4)92(1)∗}	92(1)
LFSR$_9$	Key_{10}	{∗109(2)108(12)107(23)106(35)105(32)104(35)103(31)102(25)101(22)100(10)99(10)98(7)97(2)96(9)91(1)∗}	91(1)
LFSR$_9$	Key_{11}	{∗108(9)107(15)106(31)105(38)104(42)103(31)102(26)101(21)100(15)99(10)98(4)97(3)96(4)95(3)94(2)91(1)87(1)∗}	87(1)
LFSR$_9$	Key_{12}	{∗110(1)109(1)108(8)107(24)106(41)105(34)104(48)103(19)102(34)101(21)100(9)99(6)98(5)97(3)95(2)∗}	95(2)
LFSR$_9$	Key_{13}	{∗109(5)108(8)107(26)106(32)105(30)104(40)103(38)102(31)101(14)100(13)99(4)98(5)97(5)96(2)95(1)94(1)93(1)∗}	93(1)
LFSR$_9$	Key_{14}	{∗110(1)109(3)108(12)107(18)106(30)105(51)104(28)103(38)102(26)101(15)100(14)99(3)98(5)97(6)96(3)94(1)93(1)85(1)∗}	85(1)
LFSR$_9$	Key_{15}	{∗109(3)108(10)107(20)106(39)105(28)104(35)103(43)102(27)101(16)100(12)99(7)98(5)97(5)96(3)95(1)92(1)91(1)∗}	91(1)
LFSR$_9$	Key_{16}	{∗109(3)108(6)107(15)106(37)105(38)104(45)103(38)102(16)101(25)100(17)99(5)98(3)97(3)96(2)94(1)93(2)∗}	93(2)
LFSR$_9$	Key_{17}	{∗110(1)109(2)108(11)107(20)106(32)105(37)104(42)103(34)102(29)101(19)100(8)99(6)98(5)97(2)96(3)95(4)93(1)∗}	93(1)
LFSR$_9$	Key_{18}	{∗109(1)108(8)107(22)106(32)105(40)104(42)103(34)102(21)101(22)100(8)99(9)98(8)97(5)96(2)94(1)91(1)∗}	91(1)
LFSR$_9$	Key_{19}	{∗110(1)109(4)108(5)107(20)106(40)105(28)104(47)103(36)102(23)101(17)100(12)99(8)98(9)97(2)96(1)95(1)93(1)92(1)∗}	92(1)
LFSR$_9$	Key_{20}	{∗109(2)108(15)107(32)106(37)105(36)104(30)103(24)102(25)101(20)100(9)99(8)98(8)97(4)96(3)95(2)93(1)∗}	93(1)
LFSR$_9$	Key_{21}	{∗108(10)107(18)106(29)105(49)104(41)103(36)102(23)101(15)100(11)99(9)98(6)97(4)96(1)95(3)92(1)∗}	92(1)
LFSR$_9$	Key_{22}	{∗110(1)109(4)108(14)107(13)106(26)105(38)104(38)103(42)102(27)101(20)100(11)99(9)98(5)97(2)96(3)95(1)94(1)93(1)∗}	93(1)
LFSR$_9$	Key_{23}	{∗109(1)108(10)107(18)106(28)105(38)104(44)103(34)102(30)101(22)100(13)99(4)98(6)97(5)96(3)∗}	96(3)
LFSR$_9$	Key_{24}	{∗109(2)108(10)107(18)106(32)105(38)104(39)103(33)102(26)101(17)100(18)99(5)98(5)97(6)96(2)95(1)94(3)87(1)∗}	87(1)
LFSR$_9$	Key_{25}	{∗110(1)109(4)108(14)107(21)106(30)105(33)104(40)103(34)102(26)101(12)100(19)99(9)98(4)97(5)96(2)94(1)93(1)∗}	93(1)
LFSR$_9$	Key_{26}	{∗109(2)108(2)107(24)106(37)105(45)104(36)103(35)102(26)101(16)100(14)99(4)98(7)97(2)96(2)94(3)93(1)∗}	93(1)
LFSR$_9$	Key_{27}	{∗110(1)109(2)108(9)107(18)106(32)105(43)104(40)103(34)102(26)101(18)100(20)99(7)98(1)97(2)96(2)91(1)∗}	91(1)
LFSR$_9$	Key_{28}	{∗109(5)108(10)107(22)106(24)105(38)104(46)103(33)102(21)101(17)100(11)99(7)98(8)97(5)96(3)95(3)94(1)93(2)∗}	93(2)
LFSR$_9$	Key_{29}	{∗110(2)109(4)108(11)107(23)106(24)105(35)104(37)103(44)102(25)101(18)100(13)99(10)98(4)97(4)95(1)94(1)∗}	94(1)
LFSR$_{10}$	Key_0	{∗110(1)109(4)108(8)107(26)106(34)105(41)104(32)103(34)102(23)101(26)100(13)99(2)98(4)97(1)96(3)95(1)94(1)93(1)91(1)∗}	91(1)
LFSR$_{10}$	Key_1	{∗109(3)108(8)107(19)106(30)105(42)104(44)103(33)102(21)101(20)100(14)99(6)98(7)97(4)95(5)∗}	95(5)
LFSR$_{10}$	Key_2	{∗109(3)108(9)107(23)106(29)105(39)104(45)103(28)102(31)101(19)100(12)99(6)98(3)97(4)96(4)93(1)∗}	93(1)
LFSR$_{10}$	Key_3	{∗109(1)108(9)107(21)106(38)105(39)104(31)103(23)102(24)101(23)100(9)99(14)98(6)97(9)96(5)95(1)94(1)93(2)∗}	93(2)
LFSR$_{10}$	Key_4	{∗109(5)108(7)107(20)106(25)105(37)104(36)103(41)102(24)101(24)100(11)99(7)98(5)97(9)96(3)95(1)91(1)∗}	91(1)
LFSR$_{10}$	Key_5	{∗109(3)108(11)107(19)106(35)105(43)104(37)103(38)102(30)101(10)100(14)99(9)98(3)97(1)96(2)95(1)∗}	95(1)
LFSR$_{10}$	Key_6	{∗110(1)109(5)108(12)107(17)106(34)105(39)104(36)103(37)102(25)101(14)100(16)99(5)98(4)97(2)96(4)95(3)92(1)89(1)∗}	89(1)
LFSR$_{10}$	Key_7	{∗109(4)108(8)107(20)106(33)105(37)104(38)103(30)102(26)101(21)100(7)99(10)98(10)97(5)96(1)95(4)94(1)92(1)∗}	92(1)
LFSR$_{10}$	Key_8	{∗109(2)108(5)107(26)106(34)105(40)104(34)103(43)102(29)101(16)100(10)99(6)98(3)97(3)96(2)95(3)∗}	95(3)

LFSR_{10}	Key_{9}	{∗109(3)108(10)107(23)106(35)105(30)104(44)103(29)102(29)101(20)100(7)99(9)98(4)97(5)96(4)95(1)94(1)93(1)91(1)∗}	91(1)
LFSR_{10}	Key_{10}	{∗110(1)109(2)108(11)107(24)106(33)105(40)104(42)103(40)102(24)101(11)100(14)99(6)98(2)97(1)96(2)94(2)91(1)∗}	91(1)
LFSR_{10}	Key_{11}	{∗109(6)108(10)107(17)106(25)105(38)104(40)103(38)102(37)101(17)100(13)99(3)98(8)97(2)96(2)∗}	96(2)
LFSR_{10}	Key_{12}	{∗109(2)108(9)107(20)106(29)105(42)104(35)103(40)102(24)101(19)100(15)99(10)98(2)97(2)96(4)94(2)93(1)∗}	93(1)
LFSR_{10}	Key_{13}	{∗110(1)108(11)107(21)106(33)105(34)104(40)103(43)102(22)101(18)100(14)99(5)98(5)97(3)96(2)95(3)94(1)∗}	94(1)
LFSR_{10}	Key_{14}	{∗111(1)110(1)109(1)108(2)107(29)106(38)105(43)104(37)103(36)102(22)101(16)100(15)99(7)98(4)97(1)96(1)92(1)88(1)∗}	88(1)
LFSR_{10}	Key_{15}	{∗110(1)109(1)108(8)107(18)106(32)105(31)104(41)103(35)102(26)101(26)100(10)99(4)98(12)97(4)96(2)95(3)94(1)89(1)∗}	89(1)
LFSR_{10}	Key_{16}	{∗110(1)109(2)108(9)107(27)106(28)105(33)104(38)103(42)102(20)101(19)100(16)99(8)98(4)97(1)96(2)95(5)93(1)∗}	93(1)
LFSR_{10}	Key_{17}	{∗109(2)108(10)107(26)106(33)105(36)104(39)103(34)102(31)101(20)100(9)99(5)98(2)97(5)96(2)94(1)91(1)∗}	91(1)
LFSR_{10}	Key_{18}	{∗109(5)108(10)107(26)106(33)105(41)104(42)103(33)102(17)101(18)100(10)99(7)98(7)97(3)96(4)∗}	96(4)
LFSR_{10}	Key_{19}	{∗110(3)108(15)107(25)106(27)105(41)104(39)103(36)102(25)101(17)100(11)99(7)98(4)97(1)96(2)95(2)94(1)∗}	94(1)
LFSR_{10}	Key_{20}	{∗110(1)109(2)108(7)107(22)106(30)105(48)104(37)103(27)102(31)101(17)100(8)99(14)98(5)97(4)96(2)94(1)∗}	94(1)
LFSR_{10}	Key_{21}	{∗109(7)108(8)107(16)106(37)105(39)104(38)103(36)102(20)101(21)100(11)99(8)98(6)97(1)96(3)95(2)94(2)91(1)∗}	91(1)
LFSR_{10}	Key_{22}	{∗109(2)108(12)107(23)106(34)105(38)104(33)103(32)102(22)101(24)100(18)99(6)98(6)97(1)96(3)95(2)∗}	95(2)
LFSR_{10}	Key_{23}	{∗110(1)109(2)108(10)107(23)106(30)105(36)104(32)103(34)102(27)101(14)100(21)99(12)98(7)97(3)96(1)95(2)94(1)∗}	94(1)
LFSR_{10}	Key_{24}	{∗108(14)107(20)106(39)105(42)104(41)103(31)102(22)101(16)100(9)99(13)98(5)97(2)96(1)90(1)∗}	90(1)
LFSR_{10}	Key_{25}	{∗109(2)108(7)107(20)106(37)105(39)104(38)103(29)102(23)101(25)100(15)99(9)98(7)96(3)94(2)∗}	94(2)
LFSR_{10}	Key_{26}	{∗109(2)108(10)107(21)106(28)105(43)104(36)103(33)102(28)101(14)100(18)99(3)98(9)97(4)96(3)95(2)94(1)90(1)∗}	90(1)
LFSR_{10}	Key_{27}	{∗111(1)110(2)109(2)108(10)107(18)106(28)105(46)104(37)103(35)102(22)101(20)100(13)99(6)98(8)97(3)96(2)95(1)94(1)93(1)∗}	93(1)
LFSR_{10}	Key_{28}	{∗109(4)108(16)107(23)106(28)105(34)104(36)103(36)102(23)101(23)100(16)99(6)98(1)97(6)96(2)95(1)94(1)∗}	94(1)
LFSR_{10}	Key_{29}	{∗110(2)109(1)108(18)107(23)106(32)105(38)104(41)103(28)102(26)101(19)100(10)99(10)98(3)97(2)96(2)95(1)∗}	95(1)
LFSR_{11}	Key_{0}	{∗110(2)109(5)108(8)107(25)106(29)105(33)104(41)103(35)102(22)101(21)100(10)99(7)98(9)97(5)96(1)95(2)93(1)∗}	93(1)
LFSR_{11}	Key_{1}	{∗109(3)108(7)107(35)106(26)105(35)104(37)103(38)102(29)101(14)100(11)99(4)98(5)97(2)96(4)95(3)94(2)90(1)∗}	90(1)
LFSR_{11}	Key_{2}	{∗110(1)109(4)108(17)107(18)106(18)105(41)104(41)103(39)102(25)101(19)100(7)99(9)98(3)97(9)96(2)94(3)∗}	94(3)
LFSR_{11}	Key_{3}	{∗109(1)108(7)107(19)106(35)105(42)104(42)103(27)102(32)101(18)100(13)99(7)98(5)97(8)∗}	97(8)
LFSR_{11}	Key_{4}	{∗108(8)107(13)106(33)105(43)104(42)103(31)102(22)101(24)100(11)99(8)98(11)97(4)96(2)95(3)92(1)∗}	92(1)
LFSR_{11}	Key_{5}	{∗110(1)109(2)108(4)107(20)106(33)105(39)104(34)103(35)102(36)101(20)100(12)99(6)98(5)97(4)96(3)95(1)91(1)∗}	91(1)
LFSR_{11}	Key_{6}	{∗110(2)109(2)108(5)107(26)106(27)105(40)104(36)103(31)102(24)101(20)100(18)99(11)98(7)97(4)94(2)93(1)∗}	93(1)
LFSR_{11}	Key_{7}	{∗109(4)108(14)107(22)106(29)105(39)104(46)103(18)102(20)101(27)100(11)99(9)98(5)97(7)96(3)95(1)94(1)∗}	94(1)
LFSR_{11}	Key_{8}	{∗109(2)108(10)107(21)106(38)105(40)104(44)103(30)102(16)101(16)100(16)99(8)98(6)97(2)96(1)95(4)92(2)∗}	92(2)
LFSR_{11}	Key_{9}	{∗109(2)108(15)107(22)106(29)105(34)104(38)103(36)102(21)101(18)100(14)99(11)98(3)97(9)96(3)95(1)∗}	95(1)
LFSR_{11}	Key_{10}	{∗109(1)108(15)107(21)106(31)105(37)104(35)103(48)102(28)101(11)100(13)99(9)98(3)97(2)96(1)94(1)∗}	94(1)
LFSR_{11}	Key_{11}	{∗110(2)109(7)108(9)107(19)106(33)105(35)104(31)103(29)102(31)101(21)100(15)99(11)98(6)97(3)96(2)95(2)∗}	95(2)
LFSR_{11}	Key_{12}	{∗109(4)108(12)107(22)106(33)105(39)104(33)103(35)102(27)101(21)100(13)99(7)98(4)97(2)96(1)95(1)92(1)90(1)∗}	90(1)
LFSR_{11}	Key_{13}	{∗110(1)109(5)108(10)107(17)106(28)105(44)104(27)103(42)102(34)101(17)100(16)99(7)98(6)97(2)∗}	97(2)
LFSR_{11}	Key_{14}	{∗110(1)109(2)108(8)107(25)106(41)105(36)104(45)103(35)102(26)101(12)100(11)99(10)98(2)95(1)94(1)∗}	94(1)
LFSR_{11}	Key_{15}	{∗110(1)109(1)108(7)107(22)106(39)105(41)104(35)103(39)102(19)101(15)100(16)99(12)98(4)97(1)96(2)95(1)94(1)∗}	94(1)
LFSR_{11}	Key_{16}	{∗110(1)108(6)107(24)106(36)105(43)104(39)103(35)102(25)101(11)100(15)99(10)98(5)97(3)95(1)94(1)91(1)∗}	91(1)
LFSR_{11}	Key_{17}	{∗109(1)108(9)107(17)106(32)105(34)104(43)103(37)102(29)101(12)100(16)99(10)98(4)97(4)96(2)95(2)94(1)93(2)89(1)∗}	89(1)
LFSR_{11}	Key_{18}	{∗109(1)108(5)107(24)106(37)105(43)104(34)103(36)102(26)101(13)100(14)99(12)98(6)97(3)96(1)95(1)∗}	95(1)
LFSR_{11}	Key_{19}	{∗109(1)108(7)107(24)106(41)105(31)104(44)103(28)102(27)101(13)100(13)99(10)98(7)97(4)96(3)95(1)94(1)93(1)∗}	93(1)
LFSR_{11}	Key_{20}	{∗109(2)108(12)107(22)106(27)105(46)104(34)103(35)102(27)101(21)100(16)99(3)98(1)97(4)96(3)95(1)94(2)∗}	94(2)
LFSR_{11}	Key_{21}	{∗109(2)108(8)107(27)106(27)105(42)104(43)103(43)102(23)101(17)100(11)99(3)98(1)97(4)96(2)95(2)94(1)∗}	94(1)
LFSR_{11}	Key_{22}	{∗109(1)108(13)107(22)106(32)105(30)104(45)103(38)102(21)101(19)100(10)99(9)98(7)97(4)96(3)95(1)93(1)∗}	93(1)
LFSR_{11}	Key_{23}	{∗109(6)108(7)107(16)106(36)105(47)104(46)103(23)102(27)101(14)100(10)99(6)98(8)97(5)95(1)94(2)93(2)∗}	93(2)
LFSR_{11}	Key_{24}	{∗110(1)109(1)108(9)107(17)106(36)105(42)104(38)103(31)102(21)101(19)100(19)99(5)98(8)97(2)96(1)95(4)94(1)93(1)∗}	93(1)
LFSR_{11}	Key_{25}	{∗109(4)108(8)107(18)106(26)105(43)104(46)103(31)102(34)101(15)100(10)99(10)98(6)96(2)95(2)93(1)∗}	93(1)
LFSR_{11}	Key_{26}	{∗109(5)108(11)107(27)106(35)105(34)104(34)103(32)102(27)101(17)100(12)99(8)98(4)97(4)96(1)95(1)94(2)93(2)∗}	93(2)
LFSR_{11}	Key_{27}	{∗110(2)109(4)108(8)107(18)106(32)105(44)104(47)103(34)102(24)101(12)100(8)99(7)98(8)97(6)96(1)93(1)∗}	93(1)
LFSR_{11}	Key_{28}	{∗109(6)108(12)107(29)106(25)105(41)104(36)103(34)102(24)101(22)100(8)99(9)98(5)97(1)96(1)94(1)93(2)∗}	93(2)
LFSR_{11}	Key_{29}	{∗109(4)108(10)107(17)106(29)105(34)104(51)103(39)102(17)101(17)100(16)99(8)98(5)97(5)96(1)95(2)93(1)∗}	93(1)
LFSR_{12}	Key_{0}	{∗109(1)108(5)107(25)106(28)105(33)104(48)103(29)102(29)101(21)100(17)99(13)98(2)96(1)95(1)94(2)93(1)∗}	93(1)
LFSR_{12}	Key_{1}	{∗109(4)108(11)107(20)106(27)105(42)104(34)103(42)102(27)101(12)100(7)99(11)98(10)97(2)96(2)95(1)94(2)92(1)91(1)∗}	91(1)
LFSR_{12}	Key_{2}	{∗110(1)109(3)108(8)107(13)106(28)105(43)104(32)103(41)102(26)101(16)100(15)99(16)98(6)97(2)96(3)94(2)93(1)∗}	93(1)
LFSR_{12}	Key_{3}	{∗110(1)109(2)108(8)107(19)106(28)105(34)104(42)103(38)102(28)101(19)100(18)99(6)98(7)96(4)95(2)∗}	95(2)
LFSR_{12}	Key_{4}	{∗110(2)109(3)108(4)107(18)106(26)105(35)104(40)103(40)102(37)101(24)100(9)99(9)98(2)97(1)96(3)95(2)94(1)∗}	94(1)
LFSR_{12}	Key_{5}	{∗109(6)108(14)107(19)106(34)105(42)104(29)103(33)102(25)101(15)100(8)99(17)98(6)97(4)96(1)95(2)92(1)∗}	92(1)
LFSR_{12}	Key_{6}	{∗110(1)109(4)108(13)107(12)106(36)105(40)104(46)103(25)102(25)101(16)100(14)99(8)98(5)97(4)96(3)95(3)94(1)∗}	94(1)
LFSR_{12}	Key_{7}	{∗109(2)108(7)107(25)106(30)105(33)104(34)103(31)102(31)101(23)100(11)99(16)98(4)97(5)96(2)94(2)∗}	94(2)
LFSR_{12}	Key_{8}	{∗110(3)109(3)108(8)107(23)106(32)105(43)104(32)103(24)102(23)101(22)100(10)99(11)98(8)97(7)96(1)95(3)94(1)93(2)∗}	93(2)
LFSR_{12}	Key_{9}	{∗109(6)108(14)107(22)106(29)105(37)104(42)103(21)102(34)101(16)100(16)99(4)98(6)97(6)96(2)93(1)∗}	93(1)
LFSR_{12}	Key_{10}	{∗109(4)108(5)107(21)106(24)105(36)104(39)103(37)102(27)101(26)100(13)99(7)98(8)97(4)96(3)95(1)94(1)∗}	94(1)
LFSR_{12}	Key_{11}	{∗109(4)108(14)107(30)106(25)105(39)104(35)103(28)102(22)101(27)100(14)99(8)98(5)97(2)96(2)95(1)∗}	95(1)

LFSR_{12}	Key_{12}	{∗109(4)108(6)107(26)106(34)105(28)104(38)103(36)102(26)101(13)100(19)99(12)98(3)97(5)96(3)95(1)94(1)93(1)∗}	93(1)
LFSR_{12}	Key_{13}	{∗109(4)108(7)107(24)106(40)105(35)104(37)103(37)102(15)101(13)100(13)99(15)98(8)97(3)96(3)95(1)92(1)∗}	92(1)
LFSR_{12}	Key_{14}	{∗110(1)109(1)108(9)107(23)106(28)105(38)104(35)103(28)102(29)101(18)100(16)99(6)98(11)97(3)96(4)94(2)93(1)92(1)91(1)90(1)∗}	90(1)
LFSR_{12}	Key_{15}	{∗109(5)108(9)107(16)106(33)105(44)104(40)103(39)102(23)101(18)100(12)99(10)98(2)97(2)94(2)93(1)∗}	93(1)
LFSR_{12}	Key_{16}	{∗110(1)109(5)108(9)107(25)106(45)105(38)104(25)103(35)102(25)101(12)100(14)99(8)98(6)97(1)96(2)95(1)94(2)93(1)92(1)∗}	92(1)
LFSR_{12}	Key_{17}	{∗110(2)109(2)108(10)107(19)106(31)105(44)104(35)103(27)102(29)101(19)100(14)99(9)98(9)97(3)96(3)∗}	96(3)
LFSR_{12}	Key_{18}	{∗110(2)109(5)108(10)107(18)106(34)105(36)104(41)103(38)102(28)101(15)100(8)99(9)98(5)97(4)96(1)95(1)93(1)∗}	93(1)
LFSR_{12}	Key_{19}	{∗109(1)108(10)107(27)106(34)105(25)104(34)103(41)102(31)101(16)100(11)99(7)98(8)97(3)96(5)95(2)93(1)∗}	93(1)
LFSR_{12}	Key_{20}	{∗109(1)108(6)107(26)106(38)105(32)104(33)103(36)102(27)101(21)100(19)99(6)98(5)97(2)96(3)95(1)∗}	95(1)
LFSR_{12}	Key_{21}	{∗110(2)109(6)108(7)107(20)106(35)105(33)104(38)103(33)102(22)101(23)100(10)99(13)98(4)97(3)96(3)95(2)94(1)93(1)∗}	93(1)
LFSR_{12}	Key_{22}	{∗109(1)108(22)107(20)106(26)105(44)104(37)103(33)102(22)101(9)100(7)99(17)98(3)97(8)96(3)95(2)93(1)90(1)∗}	90(1)
LFSR_{12}	Key_{23}	{∗109(3)108(12)107(17)106(33)105(45)104(35)103(35)102(27)101(18)100(9)99(8)98(6)97(3)96(4)93(1)∗}	93(1)
LFSR_{12}	Key_{24}	{∗109(3)108(15)107(19)106(30)105(49)104(35)103(36)102(23)101(17)100(11)99(6)98(3)97(2)96(4)95(2)93(1)∗}	93(1)
LFSR_{12}	Key_{25}	{∗110(1)109(2)108(9)107(19)106(41)105(40)104(29)103(30)102(30)101(17)100(12)99(15)98(5)97(3)96(1)94(1)91(1)∗}	91(1)
LFSR_{12}	Key_{26}	{∗109(3)108(10)107(23)106(33)105(38)104(43)103(43)102(20)101(15)100(15)99(4)98(5)97(2)93(2)∗}	93(2)
LFSR_{12}	Key_{27}	{∗109(5)108(8)107(22)106(31)105(43)104(36)103(35)102(16)101(15)100(20)99(8)98(3)97(6)96(2)95(4)94(2)∗}	94(2)
LFSR_{12}	Key_{28}	{∗109(3)108(5)107(25)106(39)105(38)104(34)103(37)102(30)101(13)100(9)99(9)98(6)97(6)96(2)∗}	96(2)
LFSR_{12}	Key_{29}	{∗109(3)108(8)107(19)106(35)105(40)104(39)103(35)102(29)101(21)100(12)99(4)98(3)97(6)96(1)94(1)∗}	94(1)
LFSR_{13}	Key_{0}	{∗109(1)108(4)107(25)106(32)105(37)104(33)103(37)102(32)101(17)100(14)99(14)98(5)97(3)93(2)∗}	93(2)
LFSR_{13}	Key_{1}	{∗109(2)108(7)107(23)106(24)105(42)104(44)103(40)102(21)101(22)100(12)99(11)98(4)97(2)94(1)91(1)∗}	91(1)
LFSR_{13}	Key_{2}	{∗110(1)109(2)108(7)107(21)106(42)105(29)104(42)103(32)102(22)101(18)100(14)99(10)98(6)97(3)96(3)94(3)93(1)∗}	93(1)
LFSR_{13}	Key_{3}	{∗109(3)108(9)107(19)106(38)105(34)104(40)103(33)102(21)101(20)100(16)99(7)98(9)97(1)96(2)95(3)93(1)∗}	93(1)
LFSR_{13}	Key_{4}	{∗109(2)108(14)107(22)106(33)105(33)104(40)103(38)102(21)101(21)100(14)99(5)98(4)97(3)96(5)92(1)∗}	92(1)
LFSR_{13}	Key_{5}	{∗109(5)108(10)107(27)106(42)105(35)104(38)103(32)102(22)101(13)100(12)99(8)98(3)97(2)96(4)95(2)93(1)∗}	93(1)
LFSR_{13}	Key_{6}	{∗109(2)108(8)107(27)106(31)105(40)104(50)103(31)102(13)101(18)100(13)99(9)98(6)97(2)96(2)95(3)94(1)∗}	94(1)
LFSR_{13}	Key_{7}	{∗109(3)108(13)107(13)106(41)105(35)104(39)103(29)102(22)101(19)100(15)99(12)97(5)96(3)95(4)92(3)∗}	92(3)
LFSR_{13}	Key_{8}	{∗109(2)108(13)107(18)106(40)105(39)104(37)103(32)102(27)101(11)100(11)99(5)98(8)97(1)96(6)95(3)94(1)93(1)86(1)∗}	86(1)
LFSR_{13}	Key_{9}	{∗109(3)108(13)107(20)106(36)105(42)104(31)103(37)102(28)101(19)100(12)99(7)98(2)97(2)96(3)94(1)∗}	94(1)
LFSR_{13}	Key_{10}	{∗109(1)108(12)107(27)106(29)105(48)104(35)103(31)102(20)101(21)100(13)99(8)98(3)97(4)94(2)92(1)91(1)∗}	91(1)
LFSR_{13}	Key_{11}	{∗109(2)108(11)107(22)106(31)105(41)104(31)103(35)102(18)101(27)100(12)99(7)98(9)97(2)96(3)95(3)94(1)89(1)∗}	89(1)
LFSR_{13}	Key_{12}	{∗109(1)108(14)107(21)106(36)105(44)104(40)103(28)102(29)101(13)100(11)99(9)98(2)97(3)96(3)95(1)94(1)∗}	94(1)
LFSR_{13}	Key_{13}	{∗109(1)108(7)107(18)106(38)105(47)104(41)103(41)102(18)101(14)100(12)99(6)98(4)97(2)96(1)94(2)93(2)91(2)∗}	91(2)
LFSR_{13}	Key_{14}	{∗110(1)109(2)108(9)107(22)106(23)105(39)104(48)103(28)102(33)101(18)100(13)99(10)98(3)97(2)96(2)95(1)92(1)90(1)∗}	90(1)
LFSR_{13}	Key_{15}	{∗109(3)108(6)107(24)106(33)105(45)104(33)103(38)102(22)101(14)100(12)99(8)98(7)97(3)96(5)95(1)94(2)∗}	94(2)
LFSR_{13}	Key_{16}	{∗110(1)109(3)108(13)107(16)106(38)105(41)104(39)103(26)102(24)101(23)100(11)99(8)98(3)97(3)96(2)95(2)94(2)92(1)∗}	92(1)
LFSR_{13}	Key_{17}	{∗109(2)108(15)107(25)106(26)105(34)104(35)103(26)102(27)101(27)100(17)99(8)98(3)97(4)96(6)90(1)∗}	90(1)
LFSR_{13}	Key_{18}	{∗110(1)109(5)108(8)107(21)106(29)105(47)104(41)103(27)102(19)101(15)100(17)99(11)98(3)97(6)96(5)94(1)∗}	94(1)
LFSR_{13}	Key_{19}	{∗109(2)108(12)107(25)106(31)105(32)104(36)103(35)102(24)101(22)100(18)99(6)98(3)97(3)96(4)95(2)94(1)∗}	94(1)
LFSR_{13}	Key_{20}	{∗109(6)108(7)107(26)106(38)105(42)104(42)103(31)102(28)101(10)100(9)99(8)98(3)97(5)96(1)∗}	96(1)
LFSR_{13}	Key_{21}	{∗110(1)109(3)108(7)107(19)106(28)105(41)104(43)103(31)102(33)101(20)100(13)99(5)98(5)97(6)96(1)∗}	96(1)
LFSR_{13}	Key_{22}	{∗110(1)109(5)108(9)107(25)106(37)105(32)104(47)103(35)102(16)101(25)100(8)99(7)98(3)97(3)96(2)94(1)∗}	94(1)
LFSR_{13}	Key_{23}	{∗109(3)108(13)107(17)106(35)105(40)104(36)103(31)102(23)101(23)100(13)99(12)98(5)97(1)96(1)95(1)94(1)93(1)∗}	93(1)
LFSR_{13}	Key_{24}	{∗109(2)108(4)107(27)106(31)105(38)104(36)103(32)102(32)101(18)100(14)99(10)98(4)97(2)96(1)95(3)93(1)92(1)∗}	92(1)
LFSR_{13}	Key_{25}	{∗110(1)108(10)107(28)106(32)105(29)104(43)103(30)102(25)101(20)100(13)99(6)98(8)97(4)96(3)95(2)93(1)92(1)∗}	92(1)
LFSR_{13}	Key_{26}	{∗110(1)109(2)108(15)107(24)106(37)105(40)104(36)103(31)102(26)101(18)100(8)99(6)98(4)97(3)96(3)94(1)90(1)∗}	90(1)
LFSR_{13}	Key_{27}	{∗110(1)109(5)108(7)107(20)106(22)105(41)104(29)103(36)102(35)101(18)100(17)99(4)98(9)97(4)96(6)94(1)93(1)∗}	93(1)
LFSR_{13}	Key_{28}	{∗109(1)108(15)107(20)106(34)105(29)104(43)103(18)102(36)101(20)100(12)99(9)98(7)97(7)96(3)93(1)91(1)∗}	91(1)
LFSR_{13}	Key_{29}	{∗109(5)108(10)107(26)106(25)105(38)104(41)103(24)102(27)101(27)100(14)99(9)98(4)97(1)96(3)94(1)93(1)∗}	93(1)
LFSR_{14}	Key_{0}	{∗110(1)109(4)108(14)107(21)106(37)105(44)104(27)103(33)102(24)101(16)100(10)99(6)98(5)97(5)96(7)95(1)94(1)∗}	94(1)
LFSR_{14}	Key_{1}	{∗109(2)108(7)107(21)106(31)105(40)104(37)103(30)102(28)101(16)100(20)99(10)98(6)97(1)96(3)95(1)94(2)93(1)∗}	93(1)
LFSR_{14}	Key_{2}	{∗109(3)108(5)107(25)106(33)105(43)104(40)103(28)102(18)101(19)100(12)99(16)98(7)97(4)96(1)95(1)93(1)∗}	93(1)
LFSR_{14}	Key_{3}	{∗109(5)108(8)107(17)106(30)105(51)104(34)103(42)102(22)101(19)100(7)99(9)98(5)97(3)96(2)95(1)94(1)∗}	94(1)
LFSR_{14}	Key_{4}	{∗109(3)108(9)107(16)106(38)105(38)104(32)103(32)102(27)101(23)100(14)99(9)98(5)97(5)96(1)95(2)94(1)92(1)∗}	92(1)
LFSR_{14}	Key_{5}	{∗109(4)108(12)107(22)106(32)105(44)104(37)103(31)102(23)101(11)100(13)99(9)98(8)97(4)96(4)95(1)94(1)∗}	94(1)
LFSR_{14}	Key_{6}	{∗110(1)109(4)108(9)107(26)106(34)105(32)104(30)103(38)102(32)101(18)100(14)99(12)98(2)97(3)96(1)∗}	96(1)
LFSR_{14}	Key_{7}	{∗109(4)108(16)107(22)106(32)105(33)104(45)103(31)102(30)101(15)100(3)99(12)98(4)97(3)96(2)95(2)93(2)∗}	93(2)
LFSR_{14}	Key_{8}	{∗110(1)109(4)108(12)107(17)106(32)105(43)104(32)103(36)102(18)101(22)100(10)99(10)98(8)97(3)96(6)94(1)90(1)∗}	90(1)
LFSR_{14}	Key_{9}	{∗110(1)109(2)108(9)107(18)106(27)105(37)104(45)103(39)102(23)101(17)100(17)99(11)98(7)97(1)95(2)∗}	95(2)
LFSR_{14}	Key_{10}	{∗110(1)109(3)108(5)107(26)106(34)105(35)104(38)103(35)102(26)101(16)100(14)99(10)98(2)97(3)96(5)95(2)94(1)∗}	94(1)
LFSR_{14}	Key_{11}	{∗110(1)109(4)108(8)107(14)106(31)105(45)104(51)103(29)102(19)101(22)100(15)99(8)98(3)97(3)95(2)93(1)∗}	93(1)
LFSR_{14}	Key_{12}	{∗109(5)108(15)107(20)106(41)105(33)104(42)103(21)102(25)101(17)100(10)99(11)98(7)97(6)96(1)95(1)93(1)∗}	93(1)
LFSR_{14}	Key_{13}	{∗109(4)108(10)107(14)106(28)105(37)104(45)103(30)102(19)101(28)100(12)99(12)98(9)97(3)96(2)95(1)92(1)90(1)∗}	90(1)
LFSR_{14}	Key_{14}	{∗110(1)109(5)108(4)107(32)106(40)105(44)104(37)103(31)102(18)101(15)100(6)99(5)98(8)97(5)96(3)94(1)91(1)∗}	91(1)

LFSR_{14}	Key_{15}	{∗110(1)109(4)108(9)107(27)106(36)105(41)104(39)103(34)102(20)101(12)100(12)99(7)98(5)97(5)96(1)95(2)94(1)∗}	94(1)
LFSR_{14}	Key_{16}	{∗109(3)108(6)107(22)106(35)105(45)104(34)103(31)102(35)101(23)100(8)99(8)98(1)97(1)96(4)∗}	96(4)
LFSR_{14}	Key_{17}	{∗109(2)108(5)107(23)106(32)105(37)104(39)103(23)102(31)101(23)100(15)99(15)98(4)97(2)96(3)95(1)94(1)∗}	94(1)
LFSR_{14}	Key_{18}	{∗109(3)108(9)107(17)106(30)105(38)104(34)103(35)102(26)101(24)100(11)99(10)98(7)97(9)95(1)94(1)93(1)∗}	93(1)
LFSR_{14}	Key_{19}	{∗108(10)107(15)106(27)105(40)104(52)103(37)102(25)101(12)100(15)99(15)98(5)97(1)96(2)∗}	96(2)
LFSR_{14}	Key_{20}	{∗109(3)108(19)107(18)106(39)105(38)104(45)103(23)102(21)101(18)100(11)99(12)98(5)97(1)96(2)88(1)∗}	88(1)
LFSR_{14}	Key_{21}	{∗109(1)108(9)107(19)106(27)105(40)104(36)103(36)102(38)101(15)100(10)99(11)98(3)97(6)96(2)93(2)91(1)∗}	91(1)
LFSR_{14}	Key_{22}	{∗109(1)108(9)107(22)106(40)105(40)104(37)103(20)102(24)101(21)100(23)99(7)98(4)97(1)96(4)95(1)94(1)93(1)∗}	93(1)
LFSR_{14}	Key_{23}	{∗109(4)108(10)107(17)106(38)105(36)104(40)103(38)102(18)101(22)100(11)99(12)98(5)97(2)96(2)91(1)∗}	91(1)
LFSR_{14}	Key_{24}	{∗110(1)109(4)108(9)107(24)106(35)105(44)104(34)103(34)102(15)101(23)100(10)99(8)98(6)97(1)96(3)95(3)94(2)∗}	94(2)
LFSR_{14}	Key_{25}	{∗110(1)109(1)108(12)107(17)106(36)105(44)104(42)103(27)102(18)101(22)100(13)99(8)98(5)97(3)96(2)95(3)94(1)89(1)∗}	89(1)
LFSR_{14}	Key_{26}	{∗110(1)109(2)108(8)107(20)106(32)105(36)104(48)103(33)102(26)101(20)100(13)99(10)98(2)97(1)96(2)95(1)90(1)∗}	90(1)
LFSR_{14}	Key_{27}	{∗110(1)109(4)108(4)107(15)106(34)105(40)104(37)103(37)102(26)101(20)100(22)99(3)98(4)97(1)96(2)95(3)94(3)∗}	94(3)
LFSR_{14}	Key_{28}	{∗109(2)108(12)107(22)106(26)105(42)104(32)103(34)102(23)101(24)100(14)99(13)98(3)97(5)96(1)94(2)93(1)∗}	93(1)
LFSR_{14}	Key_{29}	{∗110(2)108(13)107(15)106(31)105(45)104(38)103(33)102(24)101(21)100(12)99(8)98(3)97(3)96(3)95(1)94(2)93(2)∗}	93(2)
LFSR_{15}	Key_{0}	{∗110(1)109(2)108(13)107(22)106(28)105(46)104(44)103(27)102(28)101(16)100(8)99(14)98(2)97(3)96(1)95(1)∗}	95(1)
LFSR_{15}	Key_{1}	{∗110(1)109(3)108(6)107(17)106(27)105(48)104(47)103(26)102(22)101(18)100(12)99(9)98(8)97(4)96(6)95(2)∗}	95(2)
LFSR_{15}	Key_{2}	{∗110(1)108(5)107(30)106(31)105(43)104(45)103(23)102(25)101(20)100(10)99(7)98(7)97(4)96(1)94(3)93(1)∗}	93(1)
LFSR_{15}	Key_{3}	{∗110(1)109(4)108(4)107(18)106(47)105(31)104(34)103(37)102(32)101(16)100(10)99(5)98(4)97(3)96(4)95(3)94(1)93(2)∗}	93(2)
LFSR_{15}	Key_{4}	{∗109(3)108(6)107(29)106(28)105(33)104(36)103(32)102(27)101(24)100(13)99(10)98(8)97(2)96(2)95(3)∗}	95(3)
LFSR_{15}	Key_{5}	{∗109(3)108(7)107(19)106(33)105(40)104(42)103(25)102(26)101(21)100(13)99(14)98(9)97(3)95(1)∗}	95(1)
LFSR_{15}	Key_{6}	{∗109(5)108(11)107(25)106(36)105(39)104(47)103(23)102(18)101(22)100(13)99(3)98(7)97(3)96(2)89(1)88(1)∗}	88(1)
LFSR_{15}	Key_{7}	{∗109(4)108(7)107(17)106(34)105(34)104(36)103(32)102(33)101(19)100(15)99(11)98(9)97(1)96(2)94(1)91(1)∗}	91(1)
LFSR_{15}	Key_{8}	{∗109(2)108(8)107(16)106(47)105(32)104(39)103(35)102(21)101(17)100(18)99(12)98(5)97(2)96(2)∗}	96(2)
LFSR_{15}	Key_{9}	{∗110(1)109(7)108(9)107(16)106(35)105(45)104(38)103(29)102(22)101(18)100(15)99(7)98(5)97(4)96(1)94(2)93(1)90(1)∗}	90(1)
LFSR_{15}	Key_{10}	{∗109(4)108(14)107(16)106(45)105(36)104(24)103(32)102(28)101(19)100(18)99(13)98(3)97(1)96(2)95(1)∗}	95(1)
LFSR_{15}	Key_{11}	{∗110(1)109(6)108(12)107(20)106(34)105(31)104(45)103(32)102(19)101(12)100(18)99(6)98(7)97(8)96(1)95(3)93(1)∗}	93(1)
LFSR_{15}	Key_{12}	{∗111(1)109(2)108(12)107(15)106(37)105(31)104(31)103(34)102(30)101(19)100(18)99(9)98(6)97(6)96(3)95(1)93(1)∗}	93(1)
LFSR_{15}	Key_{13}	{∗110(1)109(2)108(13)107(19)106(30)105(38)104(40)103(34)102(31)101(13)100(11)99(15)98(3)97(3)96(2)94(1)∗}	94(1)
LFSR_{15}	Key_{14}	{∗109(3)108(4)107(30)106(24)105(46)104(25)103(42)102(22)101(22)100(15)99(11)98(6)97(2)96(1)95(2)93(1)∗}	93(1)
LFSR_{15}	Key_{15}	{∗110(1)109(2)108(12)107(16)106(32)105(37)104(42)103(39)102(30)101(13)100(14)99(4)98(4)97(5)96(2)95(2)94(1)∗}	94(1)
LFSR_{15}	Key_{16}	{∗110(1)109(2)108(13)107(17)106(25)105(38)104(46)103(36)102(19)101(23)100(15)99(6)98(3)97(7)96(4)90(1)∗}	90(1)
LFSR_{15}	Key_{17}	{∗109(4)108(12)107(24)106(29)105(31)104(35)103(38)102(31)101(23)100(7)99(8)98(8)97(1)96(1)95(1)94(1)92(2)∗}	92(2)
LFSR_{15}	Key_{18}	{∗109(4)108(12)107(23)106(30)105(33)104(39)103(29)102(29)101(29)100(14)99(3)98(5)97(2)96(3)94(1)∗}	94(1)
LFSR_{15}	Key_{19}	{∗109(3)108(6)107(28)106(34)105(33)104(32)103(39)102(21)101(25)100(17)99(9)98(5)97(1)96(1)95(2)∗}	95(2)
LFSR_{15}	Key_{20}	{∗110(1)109(1)108(6)107(16)106(24)105(36)104(47)103(47)102(26)101(21)100(13)99(5)98(3)97(5)96(4)95(1)∗}	95(1)
LFSR_{15}	Key_{21}	{∗110(1)109(2)108(10)107(25)106(35)105(35)104(36)103(36)102(24)101(18)100(14)99(4)98(7)97(3)96(4)93(1)90(1)∗}	90(1)
LFSR_{15}	Key_{22}	{∗110(2)109(2)108(8)107(19)106(37)105(34)104(47)103(28)102(25)101(24)100(14)99(3)98(8)96(3)95(1)92(1)∗}	92(1)
LFSR_{15}	Key_{23}	{∗110(1)109(4)108(11)107(20)106(29)105(35)104(40)103(32)102(31)101(24)100(14)99(4)98(6)97(2)96(1)95(1)93(1)∗}	93(1)
LFSR_{15}	Key_{24}	{∗109(3)108(10)107(25)106(32)105(38)104(39)103(29)102(20)101(30)100(12)99(5)98(4)97(3)96(1)95(3)94(1)92(1)∗}	92(1)
LFSR_{15}	Key_{25}	{∗109(2)108(5)107(23)106(37)105(48)104(38)103(26)102(25)101(15)100(13)99(13)98(6)97(3)96(2)∗}	96(2)
LFSR_{15}	Key_{26}	{∗110(2)109(3)108(12)107(21)106(35)105(45)104(42)103(27)102(19)101(18)100(10)99(6)98(6)97(6)96(4)∗}	96(4)
LFSR_{15}	Key_{27}	{∗109(4)108(10)107(17)106(32)105(38)104(37)103(33)102(26)101(17)100(12)99(14)98(4)97(5)96(5)94(1)90(1)∗}	90(1)
LFSR_{15}	Key_{28}	{∗110(1)109(2)108(10)107(21)106(34)105(48)104(36)103(21)102(28)101(14)100(15)99(6)98(10)97(3)96(2)95(2)94(2)93(1)∗}	93(1)
LFSR_{15}	Key_{29}	{∗109(3)108(9)107(33)106(36)105(39)104(37)103(32)102(13)101(12)100(16)99(7)98(7)97(6)95(1)94(3)93(1)90(1)∗}	90(1)

Table 8.4: Distribution of the Nonlinearity of the component functions of AES

LFSR	Key	Multiplicity	Min
LFSR_{0}	Key_{0}	{∗108(5)107(11)106(22)105(15)104(21)103(20)102(12)101(7)100(7)99(1)98(4)97(2)96(1)∗}	96(1)
LFSR_{0}	Key_{1}	{∗109(1)108(4)107(17)106(13)105(15)104(21)103(14)102(10)101(8)100(8)99(6)98(3)97(3)96(3)95(1)94(1)∗}	94(1)
LFSR_{0}	Key_{2}	{∗108(3)107(10)106(17)105(20)104(24)103(15)102(10)101(10)100(6)99(4)98(3)97(3)96(1)94(1)92(1)∗}	92(1)
LFSR_{0}	Key_{3}	{∗109(1)108(9)107(13)106(17)105(18)104(15)103(10)102(13)101(7)100(10)99(7)98(2)97(2)95(3)93(1)∗}	93(1)
LFSR_{0}	Key_{4}	{∗109(2)108(6)107(6)106(16)105(16)104(20)103(19)102(15)101(14)100(5)99(5)98(1)97(1)96(1)95(1)∗}	95(1)
LFSR_{0}	Key_{5}	{∗109(1)108(3)107(9)106(16)105(18)104(16)103(23)102(10)101(6)100(10)99(7)98(2)97(3)96(2)95(1)94(1)∗}	94(1)
LFSR_{0}	Key_{6}	{∗109(1)108(5)107(11)106(16)105(18)104(17)103(14)102(20)101(11)100(8)99(3)98(1)96(2)91(1)∗}	91(1)
LFSR_{0}	Key_{7}	{∗109(3)108(1)107(10)106(15)105(20)104(25)103(19)102(15)101(13)100(2)99(3)97(1)91(1)∗}	91(1)
LFSR_{0}	Key_{8}	{∗109(3)108(3)107(17)106(17)105(17)104(14)103(22)102(13)101(4)100(7)99(2)98(5)97(4)∗}	97(4)
LFSR_{0}	Key_{9}	{∗108(5)107(15)106(13)105(25)104(23)103(16)102(11)101(8)100(2)99(4)98(1)97(3)96(1)94(1)∗}	94(1)
LFSR_{0}	Key_{10}	{∗109(1)108(5)107(19)106(15)105(20)104(8)103(22)102(9)101(11)100(6)99(5)98(4)97(2)93(1)∗}	93(1)
LFSR_{0}	Key_{11}	{∗109(1)108(3)107(11)106(13)105(25)104(20)103(13)102(18)101(7)100(8)99(5)98(1)97(1)96(1)92(1)∗}	92(1)

LFSR$_0$	Key_{12}	{*110(1)109(1)108(3)107(12)106(16)105(24)104(22)103(13)102(11)101(7)100(9)99(4)98(2)97(2)96(1)*}	96(1)
LFSR$_0$	Key_{13}	{*109(2)108(3)107(11)106(16)105(22)104(19)103(18)102(11)101(5)100(5)99(6)98(4)97(2)96(1)94(3)*}	94(3)
LFSR$_0$	Key_{14}	{*109(2)108(2)107(9)106(18)105(22)104(19)103(15)102(12)101(11)100(6)99(10)98(1)94(1)*}	94(1)
LFSR$_0$	Key_{15}	{*109(2)108(4)107(9)106(16)105(23)104(17)103(18)102(11)101(11)100(5)99(6)98(2)97(1)96(2)95(1)*}	95(1)
LFSR$_0$	Key_{16}	{*110(2)109(2)108(3)107(12)106(12)105(25)104(13)103(14)102(13)101(12)100(9)99(7)98(1)97(1)96(2)*}	96(2)
LFSR$_0$	Key_{17}	{*110(1)109(1)108(5)107(11)106(20)105(23)104(14)103(15)102(13)101(12)100(5)99(3)98(1)97(2)96(1)95(1)*}	95(1)
LFSR$_0$	Key_{18}	{*108(2)107(16)106(17)105(20)104(19)103(16)102(13)101(11)100(4)99(5)98(2)97(2)96(1)*}	96(1)
LFSR$_0$	Key_{19}	{*109(2)108(5)107(12)106(15)105(23)104(23)103(16)102(11)101(6)100(7)99(3)98(2)95(1)92(1)91(1)*}	91(1)
LFSR$_0$	Key_{20}	{*109(1)108(6)107(11)106(19)105(16)104(17)103(23)102(10)101(7)100(6)99(5)98(4)96(1)94(1)92(1)*}	92(1)
LFSR$_0$	Key_{21}	{*109(2)108(6)107(11)106(12)105(18)104(26)103(15)102(14)101(5)100(3)99(10)98(1)97(2)96(1)94(2)*}	94(2)
LFSR$_0$	Key_{22}	{*108(4)107(9)106(23)105(16)104(23)103(21)102(9)101(9)100(4)99(3)98(4)97(3)*}	97(3)
LFSR$_0$	Key_{23}	{*109(1)108(11)107(6)106(26)105(17)104(13)103(16)102(7)101(7)100(7)99(8)98(5)97(3)92(1)*}	92(1)
LFSR$_0$	Key_{24}	{*109(1)108(4)107(12)106(22)105(15)104(24)103(14)102(17)101(7)100(3)99(5)98(1)97(1)96(1)95(1)*}	95(1)
LFSR$_0$	Key_{25}	{*109(2)108(4)107(13)106(21)105(19)104(16)103(11)102(16)101(12)100(6)99(5)98(1)96(1)95(1)*}	95(1)
LFSR$_0$	Key_{26}	{*108(7)107(9)106(15)105(17)104(14)103(19)102(14)101(11)100(6)99(8)98(2)97(2)96(2)95(2)*}	95(2)
LFSR$_0$	Key_{27}	{*109(1)108(5)107(8)106(14)105(21)104(14)103(23)102(13)101(10)100(4)99(4)98(1)97(4)96(2)95(2)94(2)*}	94(2)
LFSR$_0$	Key_{28}	{*109(1)108(7)107(5)106(27)105(21)104(18)103(18)102(9)101(8)100(7)99(3)98(3)97(1)*}	97(1)
LFSR$_0$	Key_{29}	{*108(5)107(12)106(23)105(14)104(18)103(16)102(13)101(6)100(10)99(5)98(3)97(2)95(1)*}	95(1)
LFSR$_1$	Key_{0}	{*109(1)108(8)107(8)106(19)105(15)104(18)103(16)102(13)101(9)100(11)99(6)98(2)96(1)94(1)*}	94(1)
LFSR$_1$	Key_{1}	{*110(1)109(1)108(2)107(12)106(24)105(23)104(14)103(20)102(9)101(7)100(5)99(4)98(1)97(2)96(1)95(1)93(1)*}	93(1)
LFSR$_1$	Key_{2}	{*109(1)108(2)107(11)106(14)105(16)104(22)103(24)102(16)101(10)100(4)99(3)98(2)96(1)92(1)85(1)*}	85(1)
LFSR$_1$	Key_{3}	{*109(1)108(3)107(6)106(16)105(24)104(25)103(12)102(15)101(9)100(4)99(5)98(3)97(2)95(1)94(1)89(1)*}	89(1)
LFSR$_1$	Key_{4}	{*109(3)108(7)107(6)106(13)105(16)104(23)103(20)102(11)101(13)100(7)99(4)98(3)97(1)96(1)*}	96(1)
LFSR$_1$	Key_{5}	{*108(7)107(11)106(17)105(17)104(19)103(27)102(9)101(6)100(5)99(4)98(2)96(2)92(1)91(1)*}	91(1)
LFSR$_1$	Key_{6}	{*109(2)108(5)107(10)106(12)105(19)104(22)103(13)102(14)101(9)100(11)99(4)98(2)97(2)96(1)95(1)94(1)*}	94(1)
LFSR$_1$	Key_{7}	{*110(1)109(1)108(5)107(8)106(18)105(19)104(17)103(18)102(8)101(15)100(9)99(5)97(1)96(2)95(1)*}	95(1)
LFSR$_1$	Key_{8}	{*109(1)108(6)107(12)106(18)105(21)104(23)103(16)102(11)101(5)100(4)99(4)98(6)96(1)*}	96(1)
LFSR$_1$	Key_{9}	{*110(1)109(1)108(3)107(13)106(15)105(22)104(22)103(17)102(11)101(7)100(5)99(3)98(3)97(2)95(2)94(1)*}	94(1)
LFSR$_1$	Key_{10}	{*108(7)107(15)106(13)105(22)104(24)103(12)102(5)101(11)100(5)99(7)98(4)96(3)*}	96(3)
LFSR$_1$	Key_{11}	{*109(3)108(3)107(10)106(17)105(17)104(19)103(22)102(11)101(9)100(7)99(4)98(3)96(1)94(1)93(1)*}	93(1)
LFSR$_1$	Key_{12}	{*109(1)108(5)107(11)106(13)105(16)104(18)103(23)102(15)101(6)100(8)99(4)98(2)97(3)96(1)95(1)94(1)*}	94(1)
LFSR$_1$	Key_{13}	{*109(4)108(3)107(11)106(24)105(20)104(13)103(13)102(9)101(9)100(5)99(7)98(5)97(1)96(2)95(1)93(1)*}	93(1)
LFSR$_1$	Key_{14}	{*111(1)109(1)108(3)107(13)106(14)105(20)104(16)103(19)102(13)101(9)100(4)99(6)97(1)96(5)95(1)94(2)*}	94(2)
LFSR$_1$	Key_{15}	{*109(1)108(2)107(13)106(18)105(25)104(23)103(16)102(11)101(7)100(2)99(4)97(1)96(2)92(3)*}	92(3)
LFSR$_1$	Key_{16}	{*110(1)109(1)108(4)107(13)106(18)105(19)104(19)103(14)102(15)101(7)100(4)99(4)98(5)96(1)95(1)94(1)89(1)*}	89(1)
LFSR$_1$	Key_{17}	{*109(1)108(4)107(8)106(18)105(19)104(18)103(22)102(14)101(6)100(5)99(5)98(1)97(3)96(2)95(1)92(1)*}	92(1)
LFSR$_1$	Key_{18}	{*109(1)108(9)107(10)106(14)105(25)104(13)103(17)102(12)101(7)100(5)99(8)98(4)97(2)96(1)*}	96(1)
LFSR$_1$	Key_{19}	{*109(2)108(5)107(11)106(14)105(20)104(19)103(18)102(13)101(11)100(2)99(3)98(2)97(4)96(2)95(1)92(1)*}	92(1)
LFSR$_1$	Key_{20}	{*109(5)108(9)107(11)106(10)105(20)104(21)103(17)102(11)101(5)100(9)99(4)98(1)97(4)95(1)*}	95(1)
LFSR$_1$	Key_{21}	{*110(1)109(1)108(7)107(11)106(16)105(12)104(26)103(23)102(12)101(5)100(5)99(4)98(2)97(1)95(2)*}	95(2)
LFSR$_1$	Key_{22}	{*109(2)108(9)107(14)106(10)105(19)104(16)103(18)102(11)101(8)100(6)99(5)98(2)97(2)96(3)94(1)92(1)89(1)*}	89(1)
LFSR$_1$	Key_{23}	{*109(3)108(6)107(9)106(16)105(22)104(19)103(15)102(16)101(10)100(4)99(3)98(2)97(2)90(1)*}	90(1)
LFSR$_1$	Key_{24}	{*110(1)108(8)107(8)106(12)105(21)104(14)103(26)102(13)101(9)100(9)99(3)98(2)95(1)93(1)*}	93(1)
LFSR$_1$	Key_{25}	{*109(4)108(6)107(8)106(20)105(23)104(21)103(14)102(13)101(5)100(3)99(4)98(6)96(1)*}	96(1)
LFSR$_1$	Key_{26}	{*110(1)109(2)108(4)107(8)106(9)105(23)104(24)103(26)102(7)101(12)100(4)99(2)98(3)97(2)95(1)*}	95(1)
LFSR$_1$	Key_{27}	{*109(1)108(6)107(8)106(22)105(21)104(25)103(13)102(10)101(8)100(7)99(1)98(2)97(1)96(2)95(1)*}	95(1)
LFSR$_1$	Key_{28}	{*110(1)109(1)108(6)107(6)106(19)105(23)104(15)103(15)102(11)101(7)100(4)99(8)98(3)97(5)96(2)95(2)*}	95(2)
LFSR$_1$	Key_{29}	{*109(3)108(9)107(6)106(14)105(28)104(21)103(15)102(13)101(8)100(5)99(2)98(1)97(2)95(1)*}	95(1)
LFSR$_2$	Key_{0}	{*109(4)108(3)107(7)106(16)105(14)104(19)103(28)102(7)101(9)100(6)99(4)98(6)97(3)96(1)93(1)*}	93(1)
LFSR$_2$	Key_{1}	{*110(1)108(4)107(14)106(13)105(36)104(18)103(11)102(9)101(7)100(7)99(3)98(3)97(2)*}	97(2)
LFSR$_2$	Key_{2}	{*108(7)107(13)106(13)105(23)104(19)103(16)102(9)101(10)100(3)99(6)98(3)97(2)96(3)95(1)*}	95(1)
LFSR$_2$	Key_{3}	{*109(1)108(6)107(10)106(17)105(23)104(25)103(17)102(8)101(8)100(3)99(4)98(4)95(1)93(1)*}	93(1)
LFSR$_2$	Key_{4}	{*109(1)108(6)107(8)106(10)105(22)104(15)103(16)102(17)101(12)100(7)99(9)98(3)97(1)93(1)*}	93(1)
LFSR$_2$	Key_{5}	{*109(1)108(2)107(10)106(7)105(21)104(20)103(18)102(17)101(12)100(9)99(4)98(2)96(3)95(1)94(1)*}	94(1)
LFSR$_2$	Key_{6}	{*108(3)107(8)106(21)105(19)104(24)103(13)102(14)101(7)100(9)99(2)98(4)97(2)96(1)95(1)*}	95(1)
LFSR$_2$	Key_{7}	{*109(1)108(8)107(16)106(18)105(23)104(10)103(17)102(12)101(9)100(3)99(2)98(4)96(4)95(1)*}	95(1)
LFSR$_2$	Key_{8}	{*108(5)107(12)106(23)105(21)104(14)103(15)102(13)101(9)100(7)99(4)98(2)97(2)94(1)*}	94(1)
LFSR$_2$	Key_{9}	{*110(1)108(4)107(14)106(23)105(16)104(18)103(15)102(11)101(7)100(8)99(2)98(2)97(1)96(3)95(2)91(1)*}	91(1)
LFSR$_2$	Key_{10}	{*110(1)109(3)108(4)107(15)106(15)105(21)104(18)103(18)102(10)101(6)100(6)99(5)98(3)95(1)94(1)92(1)*}	92(1)
LFSR$_2$	Key_{11}	{*109(1)108(4)107(7)106(10)105(17)104(26)103(24)102(12)101(13)100(7)99(2)97(2)96(2)95(1)*}	95(1)
LFSR$_2$	Key_{12}	{*109(1)108(8)107(12)106(18)105(18)104(18)103(10)102(18)101(9)100(11)99(3)98(1)97(1)*}	97(1)
LFSR$_2$	Key_{13}	{*111(1)108(6)107(10)106(16)105(17)104(16)103(17)102(7)101(17)100(4)99(6)98(5)97(2)95(2)94(1)93(1)*}	93(1)
LFSR$_2$	Key_{14}	{*109(1)108(4)107(6)106(22)105(19)104(19)103(16)102(15)101(7)100(5)99(6)98(5)97(1)96(1)94(1)*}	94(1)

LFSR_2	Key_{15}	{∗108(9)107(15)106(14)105(24)104(19)103(16)102(12)101(6)100(3)99(6)98(2)97(1)96(1)∗}	96(1)
LFSR_2	Key_{16}	{∗109(5)108(8)107(9)106(19)105(12)104(13)103(14)102(11)101(9)100(7)99(10)98(6)97(1)96(3)94(1)∗}	94(1)
LFSR_2	Key_{17}	{∗109(1)108(3)107(15)106(16)105(25)104(23)103(17)102(9)101(8)100(5)99(4)98(1)97(1)∗}	97(1)
LFSR_2	Key_{18}	{∗109(1)108(3)107(11)106(13)105(16)104(19)103(18)102(15)101(8)100(10)99(4)98(4)96(4)95(2)∗}	95(2)
LFSR_2	Key_{19}	{∗109(1)108(6)107(10)106(23)105(15)104(19)103(15)102(14)101(8)100(5)99(5)98(1)97(2)96(2)94(2)∗}	94(2)
LFSR_2	Key_{20}	{∗109(4)108(2)107(9)106(18)105(19)104(22)103(14)102(16)101(10)100(5)99(2)98(2)97(1)96(2)95(1)94(1)∗}	94(1)
LFSR_2	Key_{21}	{∗109(1)108(8)107(7)106(20)105(20)104(12)103(21)102(14)101(10)100(4)99(4)98(4)97(1)96(1)94(1)∗}	94(1)
LFSR_2	Key_{22}	{∗109(2)108(5)107(12)106(11)105(19)104(21)103(16)102(13)101(15)100(6)99(5)98(2)95(1)∗}	95(1)
LFSR_2	Key_{23}	{∗109(1)108(3)107(11)106(17)105(18)104(12)103(16)102(15)101(10)100(8)99(9)98(2)97(3)96(1)95(1)92(1)∗}	92(1)
LFSR_2	Key_{24}	{∗109(2)108(1)107(9)106(19)105(16)104(21)103(18)102(16)101(7)100(9)99(6)98(2)97(1)94(1)∗}	94(1)
LFSR_2	Key_{25}	{∗109(1)108(1)107(12)106(15)105(26)104(17)103(23)102(8)101(7)100(10)98(4)97(3)96(1)∗}	96(1)
LFSR_2	Key_{26}	{∗109(2)108(5)107(8)106(20)105(21)104(21)103(19)102(10)101(8)100(3)99(3)98(4)97(4)∗}	97(4)
LFSR_2	Key_{27}	{∗109(1)108(7)107(11)106(19)105(20)104(17)103(14)102(16)101(10)100(1)99(4)98(1)97(2)95(3)94(2)∗}	94(2)
LFSR_2	Key_{28}	{∗110(1)109(2)108(5)107(6)106(12)105(19)104(20)103(21)102(12)101(14)100(5)99(5)98(1)97(2)95(1)94(1)93(1)∗}	93(1)
LFSR_2	Key_{29}	{∗109(1)108(3)107(8)106(13)105(25)104(25)103(14)102(6)101(13)100(8)99(5)98(3)96(2)95(2)∗}	95(2)
LFSR_3	Key_{0}	{∗109(2)108(8)107(13)106(15)105(22)104(12)103(14)102(14)101(12)100(4)99(6)98(2)97(2)96(1)93(1)∗}	93(1)
LFSR_3	Key_{1}	{∗110(1)109(2)108(3)107(11)106(20)105(27)104(19)103(13)102(11)101(6)100(3)99(6)98(1)97(3)96(1)91(1)∗}	91(1)
LFSR_3	Key_{2}	{∗109(1)108(2)107(15)106(27)105(16)104(18)103(12)102(11)101(12)100(8)99(1)98(3)96(2)∗}	96(2)
LFSR_3	Key_{3}	{∗110(1)109(3)108(6)107(8)106(12)105(22)104(22)103(14)102(10)101(12)100(4)99(5)98(6)97(2)96(1)∗}	96(1)
LFSR_3	Key_{4}	{∗109(3)108(3)107(11)106(17)105(19)104(18)103(15)102(8)101(8)100(9)99(6)98(6)97(2)96(1)95(1)92(1)∗}	92(1)
LFSR_3	Key_{5}	{∗108(8)107(11)106(12)105(20)104(24)103(18)102(15)101(8)100(4)99(3)98(2)97(1)96(1)94(1)∗}	94(1)
LFSR_3	Key_{6}	{∗109(1)108(5)107(12)106(19)105(12)104(20)103(15)102(9)101(7)100(15)99(4)98(4)97(3)96(1)93(1)∗}	93(1)
LFSR_3	Key_{7}	{∗110(1)109(3)108(2)107(13)106(20)105(22)104(20)103(17)102(3)101(13)100(8)99(1)98(1)97(1)96(1)95(2)∗}	95(2)
LFSR_3	Key_{8}	{∗109(2)108(2)107(16)106(22)105(20)104(15)103(10)102(9)101(8)100(9)99(3)98(4)97(3)96(1)95(1)94(3)∗}	94(3)
LFSR_3	Key_{9}	{∗109(2)108(2)107(12)106(13)105(22)104(19)103(21)102(7)101(5)100(9)99(9)98(3)97(2)96(2)∗}	96(2)
LFSR_3	Key_{10}	{∗109(2)108(6)107(3)106(17)105(27)104(19)103(10)102(12)101(11)100(10)99(5)98(1)97(2)96(2)94(1)∗}	94(1)
LFSR_3	Key_{11}	{∗109(1)108(4)107(15)106(23)105(16)104(23)103(11)102(11)101(10)100(6)99(2)98(1)97(2)95(2)94(1)∗}	94(1)
LFSR_3	Key_{12}	{∗110(1)109(2)108(3)107(8)106(21)105(21)104(19)103(21)102(7)101(10)100(4)99(3)98(4)97(1)95(1)94(1)93(1)∗}	93(1)
LFSR_3	Key_{13}	{∗109(1)108(5)107(13)106(15)105(14)104(14)103(25)102(13)101(10)100(7)99(4)98(2)97(2)95(3)∗}	95(3)
LFSR_3	Key_{14}	{∗109(1)108(4)107(8)106(12)105(23)104(17)103(14)102(13)101(13)100(9)99(5)98(6)97(2)96(1)∗}	96(1)
LFSR_3	Key_{15}	{∗109(2)108(4)107(6)106(17)105(17)104(20)103(26)102(10)101(8)100(9)99(4)98(4)95(1)∗}	95(1)
LFSR_3	Key_{16}	{∗109(2)108(6)107(16)106(17)105(16)104(18)103(17)102(13)101(10)100(3)99(3)98(2)97(2)96(1)94(1)92(1)∗}	92(1)
LFSR_3	Key_{17}	{∗110(2)109(1)108(4)107(10)106(12)105(17)104(25)103(14)102(14)101(7)100(9)99(8)98(4)90(1)∗}	90(1)
LFSR_3	Key_{18}	{∗109(2)108(2)107(13)106(18)105(17)104(17)103(18)102(13)101(7)100(9)99(4)98(5)97(1)96(1)94(1)∗}	94(1)
LFSR_3	Key_{19}	{∗109(1)108(6)107(6)106(20)105(12)104(20)103(16)102(13)101(12)100(10)99(5)98(2)97(5)∗}	97(5)
LFSR_3	Key_{20}	{∗109(2)108(6)107(10)106(16)105(20)104(16)103(20)102(9)101(15)100(4)99(4)98(5)96(1)∗}	96(1)
LFSR_3	Key_{21}	{∗109(2)108(1)107(12)106(14)105(15)104(15)103(15)102(14)101(11)100(5)99(9)98(5)97(3)96(3)95(3)93(1)∗}	93(1)
LFSR_3	Key_{22}	{∗110(1)109(3)108(6)107(13)106(17)105(18)104(13)103(12)102(15)101(10)100(9)99(6)98(3)97(2)∗}	97(2)
LFSR_3	Key_{23}	{∗109(1)108(6)107(11)106(14)105(25)104(19)103(13)102(14)101(12)100(7)99(2)98(3)94(1)∗}	94(1)
LFSR_3	Key_{24}	{∗109(5)108(2)107(15)106(16)105(14)104(18)103(24)102(8)101(8)100(8)99(4)98(3)97(1)96(1)90(1)∗}	90(1)
LFSR_3	Key_{25}	{∗109(2)108(2)107(6)106(18)105(17)104(31)103(13)102(15)101(6)100(10)99(4)98(1)97(1)94(1)92(1)∗}	92(1)
LFSR_3	Key_{26}	{∗109(3)108(4)107(15)106(10)105(17)104(15)103(23)102(18)101(8)100(6)99(2)98(1)97(1)96(2)95(1)94(2)∗}	94(2)
LFSR_3	Key_{27}	{∗108(1)107(15)106(19)105(14)104(21)103(21)102(9)101(12)100(5)99(4)98(3)97(2)96(1)94(1)∗}	94(1)
LFSR_3	Key_{28}	{∗109(1)108(6)107(18)106(16)105(12)104(23)103(19)102(14)101(3)100(4)99(5)98(1)97(1)96(3)95(1)92(1)∗}	92(1)
LFSR_3	Key_{29}	{∗109(2)108(4)107(6)106(19)105(25)104(26)103(11)102(12)101(8)100(5)99(4)98(4)97(2)∗}	97(2)
LFSR_4	Key_{0}	{∗109(1)108(9)107(9)106(12)105(24)104(18)103(13)102(14)101(8)100(5)99(9)98(1)97(2)96(1)95(1)92(1)∗}	92(1)
LFSR_4	Key_{1}	{∗109(4)108(8)107(6)106(21)105(18)104(22)103(7)102(9)101(9)100(12)99(5)98(2)97(1)96(2)95(2)∗}	95(2)
LFSR_4	Key_{2}	{∗109(2)108(5)107(8)106(19)105(25)104(15)103(9)102(12)101(8)100(9)99(6)98(4)97(2)95(3)92(1)∗}	92(1)
LFSR_4	Key_{3}	{∗109(5)108(4)107(8)106(17)105(25)104(23)103(13)102(12)101(8)100(6)99(3)98(1)97(2)91(1)∗}	91(1)
LFSR_4	Key_{4}	{∗109(2)108(3)107(8)106(19)105(15)104(24)103(12)102(14)101(12)100(5)99(6)98(3)97(1)96(1)95(1)93(2)∗}	93(2)
LFSR_4	Key_{5}	{∗112(1)109(3)108(6)107(10)106(21)105(22)104(10)103(17)102(8)101(9)100(7)99(3)98(6)97(1)96(3)93(1)∗}	93(1)
LFSR_4	Key_{6}	{∗109(2)108(4)107(5)106(20)105(24)104(22)103(14)102(11)101(12)100(3)99(2)98(1)97(3)96(1)95(1)94(1)93(1)90(1)∗}	90(1)
LFSR_4	Key_{7}	{∗109(2)108(2)107(13)106(24)105(22)104(12)103(14)102(10)101(10)100(4)99(6)98(4)97(1)96(2)95(1)94(1)∗}	94(1)
LFSR_4	Key_{8}	{∗109(1)108(6)107(15)106(19)105(21)104(19)103(14)102(11)101(7)100(8)99(1)98(5)97(1)∗}	97(1)
LFSR_4	Key_{9}	{∗108(4)107(15)106(9)105(20)104(23)103(20)102(10)101(7)100(4)99(6)98(4)97(1)96(2)95(2)91(1)∗}	91(1)
LFSR_4	Key_{10}	{∗109(2)108(5)107(13)106(20)105(24)104(9)103(19)102(11)101(9)100(7)99(2)98(3)97(3)95(1)∗}	95(1)
LFSR_4	Key_{11}	{∗110(1)109(1)108(3)107(14)106(15)105(15)104(17)103(15)102(17)101(12)100(1)99(9)98(1)97(5)96(1)94(1)∗}	94(1)
LFSR_4	Key_{12}	{∗109(2)108(6)107(11)106(22)105(17)104(20)103(17)102(14)101(3)100(3)99(5)98(4)97(2)96(1)94(1)∗}	94(1)
LFSR_4	Key_{13}	{∗110(1)109(1)108(3)107(12)106(18)105(11)104(16)103(20)102(16)101(10)100(5)99(7)98(2)97(1)96(1)95(3)94(1)∗}	94(1)
LFSR_4	Key_{14}	{∗110(1)109(3)108(6)107(6)106(17)105(18)104(18)103(15)102(13)101(8)100(7)99(6)98(4)97(2)96(3)95(1)∗}	95(1)
LFSR_4	Key_{15}	{∗108(4)107(13)106(14)105(21)104(21)103(17)102(15)101(10)100(4)99(3)98(2)97(2)95(2)∗}	95(2)
LFSR_4	Key_{16}	{∗109(3)108(5)107(12)106(14)105(19)104(24)103(12)102(10)101(12)100(6)99(3)98(2)97(2)96(1)95(2)94(1)∗}	94(1)
LFSR_4	Key_{17}	{∗109(2)108(5)107(9)106(24)105(16)104(20)103(14)102(7)101(8)100(9)99(5)98(4)97(1)96(2)95(1)93(1)∗}	93(1)

$LFSR_4$	Key_{18}	{∗109(1)108(2)107(5)106(24)105(15)104(13)103(20)102(16)101(8)100(10)99(2)98(3)97(3)96(4)95(1)94(1)∗}	94(1)
$LFSR_4$	Key_{19}	{∗110(1)109(1)108(8)107(10)106(11)105(23)104(17)103(17)102(14)101(8)100(7)99(3)98(1)97(3)96(3)93(1)∗}	93(1)
$LFSR_4$	Key_{20}	{∗110(2)108(7)107(15)106(15)105(20)104(13)103(16)102(5)101(11)100(14)99(5)98(2)97(2)94(1)∗}	94(1)
$LFSR_4$	Key_{21}	{∗109(1)108(3)107(6)106(14)105(24)104(19)103(14)102(14)101(12)100(8)99(2)98(4)97(3)95(4)∗}	95(4)
$LFSR_4$	Key_{22}	{∗108(5)107(6)106(27)105(22)104(16)103(16)102(8)101(12)100(3)99(2)98(2)97(4)96(1)94(1)93(2)92(1)∗}	92(1)
$LFSR_4$	Key_{23}	{∗109(2)108(6)107(8)106(19)105(18)104(17)103(23)102(13)101(6)100(5)99(4)97(5)96(1)94(1)∗}	94(1)
$LFSR_4$	Key_{24}	{∗109(1)108(6)107(11)106(18)105(22)104(14)103(16)102(17)101(10)100(3)99(3)98(2)97(3)94(1)91(1)∗}	91(1)
$LFSR_4$	Key_{25}	{∗109(3)108(3)107(15)106(13)105(22)104(17)103(16)102(11)101(7)100(5)99(3)98(4)97(2)96(1)95(3)93(2)91(1)∗}	91(1)
$LFSR_4$	Key_{26}	{∗110(1)109(2)108(4)107(11)106(23)105(21)104(16)103(15)102(10)101(8)100(8)99(4)97(1)96(1)95(1)94(1)92(1)∗}	92(1)
$LFSR_4$	Key_{27}	{∗109(4)108(6)107(13)106(17)105(19)104(17)103(15)102(12)101(9)100(7)99(3)98(2)97(1)96(1)95(2)∗}	95(2)
$LFSR_4$	Key_{28}	{∗110(2)109(1)108(6)107(6)106(10)105(23)104(19)103(17)102(13)101(14)100(4)99(6)98(4)97(2)96(1)∗}	96(1)
$LFSR_4$	Key_{29}	{∗109(1)107(12)106(17)105(19)104(17)103(13)102(21)101(11)100(8)99(4)98(3)97(2)∗}	97(2)
$LFSR_5$	Key_{0}	{∗109(2)108(3)107(10)106(20)105(17)104(17)103(30)102(10)101(6)100(4)99(5)98(1)97(2)96(1)∗}	96(1)
$LFSR_5$	Key_{1}	{∗109(1)108(3)107(14)106(15)105(20)104(21)103(14)102(7)101(11)100(9)99(5)98(2)97(3)96(2)95(1)∗}	95(1)
$LFSR_5$	Key_{2}	{∗109(2)108(3)107(6)106(22)105(20)104(21)103(18)102(10)101(8)100(3)99(5)98(2)97(4)96(3)95(1)∗}	95(1)
$LFSR_5$	Key_{3}	{∗109(2)108(1)107(7)106(16)105(19)104(20)103(19)102(18)101(11)100(7)99(4)98(2)97(1)96(1)∗}	96(1)
$LFSR_5$	Key_{4}	{∗110(1)109(2)108(4)107(10)106(20)105(14)104(20)103(17)102(9)101(9)100(12)99(4)98(1)97(2)96(2)94(1)∗}	94(1)
$LFSR_5$	Key_{5}	{∗109(2)108(6)107(10)106(14)105(16)104(24)103(10)102(11)101(9)100(10)99(5)98(7)97(1)96(1)93(1)87(1)∗}	87(1)
$LFSR_5$	Key_{6}	{∗109(2)108(4)107(10)106(12)105(22)104(15)103(13)102(17)101(14)100(7)99(5)98(3)97(3)95(1)∗}	95(1)
$LFSR_5$	Key_{7}	{∗109(3)108(3)107(8)106(19)105(20)104(20)103(18)102(8)101(8)100(14)99(5)98(1)92(1)∗}	92(1)
$LFSR_5$	Key_{8}	{∗110(1)108(4)107(16)106(19)105(21)104(18)103(17)102(11)101(5)100(7)99(1)98(5)96(1)93(1)92(1)∗}	92(1)
$LFSR_5$	Key_{9}	{∗109(1)108(8)107(12)106(20)105(24)104(25)103(9)102(11)101(4)100(7)99(3)98(3)94(1)∗}	94(1)
$LFSR_5$	Key_{10}	{∗109(1)108(9)107(13)106(14)105(17)104(22)103(12)102(10)101(8)100(7)99(7)98(2)97(2)96(2)94(1)91(1)∗}	91(1)
$LFSR_5$	Key_{11}	{∗109(3)108(5)107(9)106(15)105(21)104(17)103(11)102(17)101(7)100(8)99(8)98(2)97(3)96(2)∗}	96(2)
$LFSR_5$	Key_{12}	{∗109(1)107(9)106(12)105(18)104(24)103(15)102(18)101(8)100(9)99(4)98(5)97(1)96(1)95(2)94(1)∗}	94(1)
$LFSR_5$	Key_{13}	{∗108(8)107(15)106(15)105(24)104(17)103(12)102(11)101(10)100(4)99(9)98(2)95(1)∗}	95(1)
$LFSR_5$	Key_{14}	{∗108(5)107(10)106(16)105(21)104(20)103(17)102(12)101(7)100(10)99(3)98(2)97(3)96(1)95(1)∗}	95(1)
$LFSR_5$	Key_{15}	{∗110(1)108(3)107(9)106(13)105(14)104(21)103(21)102(11)101(7)100(11)99(4)98(5)97(2)96(2)95(1)93(3)∗}	93(3)
$LFSR_5$	Key_{16}	{∗111(1)109(2)108(2)107(11)106(12)105(17)104(19)103(18)102(24)101(6)100(5)99(5)97(1)96(2)95(2)92(1)∗}	92(1)
$LFSR_5$	Key_{17}	{∗108(1)107(14)106(24)105(19)104(25)103(13)102(13)101(7)100(5)99(4)98(1)97(1)95(1)∗}	95(1)
$LFSR_5$	Key_{18}	{∗108(1)107(13)106(19)105(13)104(27)103(14)102(17)101(7)100(5)99(4)98(5)97(2)95(1)∗}	95(1)
$LFSR_5$	Key_{19}	{∗109(3)108(3)107(11)106(24)105(12)104(21)103(14)102(13)101(8)100(7)99(5)98(3)97(2)96(1)95(1)∗}	95(1)
$LFSR_5$	Key_{20}	{∗109(1)108(9)107(9)106(17)105(16)104(15)103(17)102(16)101(11)100(3)99(4)98(7)97(1)96(1)91(1)∗}	91(1)
$LFSR_5$	Key_{21}	{∗110(1)109(2)108(4)107(9)106(17)105(16)104(26)103(17)102(9)101(12)100(5)99(4)98(1)97(2)96(1)92(1)87(1)∗}	87(1)
$LFSR_5$	Key_{22}	{∗109(2)108(4)107(6)106(11)105(13)104(21)103(22)102(12)101(9)100(9)99(5)98(5)97(5)96(3)95(1)∗}	95(1)
$LFSR_5$	Key_{23}	{∗109(2)108(7)107(10)106(13)105(13)104(12)103(19)102(14)101(14)100(14)99(4)98(3)97(2)93(1)∗}	93(1)
$LFSR_5$	Key_{24}	{∗108(5)107(8)106(15)105(14)104(28)103(23)102(16)101(7)100(2)99(2)98(5)97(1)94(1)93(1)∗}	93(1)
$LFSR_5$	Key_{25}	{∗109(1)108(7)107(6)106(19)105(23)104(20)103(7)102(15)101(12)100(8)99(5)98(2)97(1)96(1)95(1)∗}	95(1)
$LFSR_5$	Key_{26}	{∗109(2)108(5)107(12)106(14)105(16)104(19)103(15)102(12)101(11)100(4)99(9)98(5)97(2)95(1)94(1)∗}	94(1)
$LFSR_5$	Key_{27}	{∗109(1)108(5)107(12)106(17)105(19)104(17)103(11)102(17)101(8)100(9)99(8)97(2)96(1)95(1)∗}	95(1)
$LFSR_5$	Key_{28}	{∗109(2)108(4)107(10)106(21)105(19)104(16)103(16)102(13)101(8)100(5)99(6)98(5)97(3)∗}	97(3)
$LFSR_5$	Key_{29}	{∗108(6)107(12)106(15)105(23)104(15)103(19)102(12)101(7)100(10)99(3)98(3)96(1)94(1)93(1)∗}	93(1)
$LFSR_6$	Key_{0}	{∗108(4)107(7)106(15)105(21)104(12)103(22)102(18)101(7)100(10)99(5)98(2)97(2)95(1)94(1)92(1)∗}	92(1)
$LFSR_6$	Key_{1}	{∗108(6)107(15)106(17)105(22)104(16)103(10)102(15)101(10)100(7)99(2)98(3)97(2)96(2)95(1)∗}	95(1)
$LFSR_6$	Key_{2}	{∗108(5)107(6)106(9)105(19)104(21)103(19)102(8)101(14)100(14)99(7)98(3)97(1)94(1)91(1)∗}	91(1)
$LFSR_6$	Key_{3}	{∗110(1)108(5)107(9)106(24)105(26)104(14)103(15)102(7)101(11)100(4)99(4)98(2)97(2)96(1)95(1)94(1)92(1)∗}	92(1)
$LFSR_6$	Key_{4}	{∗108(6)107(11)106(15)105(19)104(20)103(18)102(8)101(12)100(5)99(4)98(4)97(4)96(2)∗}	96(2)
$LFSR_6$	Key_{5}	{∗109(2)108(8)107(12)106(17)105(14)104(20)103(15)102(14)101(8)100(4)99(4)98(6)97(2)96(1)92(1)∗}	92(1)
$LFSR_6$	Key_{6}	{∗110(1)109(1)108(3)107(13)106(14)105(26)104(13)103(17)102(10)101(7)100(14)99(7)98(2)∗}	98(2)
$LFSR_6$	Key_{7}	{∗108(3)107(8)106(16)105(16)104(11)103(24)102(14)101(15)100(6)99(5)98(6)97(1)95(2)93(1)∗}	93(1)
$LFSR_6$	Key_{8}	{∗109(2)108(3)107(12)106(18)105(26)104(16)103(20)102(8)101(9)100(4)99(5)98(2)96(2)95(1)∗}	95(1)
$LFSR_6$	Key_{9}	{∗109(3)108(8)107(9)106(21)105(24)104(18)103(12)102(10)101(11)100(2)99(1)98(1)97(2)95(1)94(2)93(2)91(1)∗}	91(1)
$LFSR_6$	Key_{10}	{∗109(1)108(3)107(14)106(14)105(17)104(22)103(14)102(14)101(10)100(5)99(5)98(3)97(2)96(2)95(1)90(1)∗}	90(1)
$LFSR_6$	Key_{11}	{∗109(3)108(3)107(6)106(14)105(20)104(22)103(19)102(14)101(6)100(5)99(2)98(4)97(4)96(4)95(1)94(1)∗}	94(1)
$LFSR_6$	Key_{12}	{∗109(2)108(5)107(11)106(19)105(16)104(21)103(16)102(17)101(6)100(4)99(6)98(1)97(1)96(2)95(1)∗}	95(1)
$LFSR_6$	Key_{13}	{∗110(1)109(2)108(2)107(10)106(14)105(23)104(19)103(21)102(19)101(6)100(3)99(2)98(1)97(3)94(1)93(1)∗}	93(1)
$LFSR_6$	Key_{14}	{∗108(2)107(6)106(23)105(17)104(23)103(18)102(17)101(5)100(6)99(7)98(1)96(2)94(1)∗}	94(1)
$LFSR_6$	Key_{15}	{∗109(2)108(9)107(8)106(9)105(18)104(18)103(13)102(12)101(12)100(8)99(8)98(4)97(3)96(1)95(1)94(1)87(1)∗}	87(1)
$LFSR_6$	Key_{16}	{∗109(3)108(4)107(5)106(13)105(23)104(26)103(20)102(9)101(11)100(4)99(5)98(3)97(2)∗}	97(2)
$LFSR_6$	Key_{17}	{∗110(1)109(1)108(9)107(8)106(19)105(22)104(18)103(19)102(12)101(7)100(7)99(4)98(1)∗}	98(1)
$LFSR_6$	Key_{18}	{∗108(8)107(9)106(10)105(22)104(20)103(20)102(14)101(10)100(5)99(6)97(1)96(1)95(1)91(1)∗}	91(1)
$LFSR_6$	Key_{19}	{∗109(2)108(5)107(10)106(12)105(20)104(23)103(14)102(14)101(13)100(5)99(6)98(4)∗}	98(4)
$LFSR_6$	Key_{20}	{∗108(1)107(14)106(13)105(23)104(22)103(14)102(15)101(9)100(6)99(3)98(3)97(1)96(3)94(1)∗}	94(1)

$LFSR_6$	Key_{21}	{*109(2)108(3)107(13)106(18)105(18)104(16)103(15)102(8)101(13)100(15)99(5)98(2)*}	98(2)
$LFSR_6$	Key_{22}	{*109(1)108(4)107(14)106(25)105(15)104(21)103(15)102(6)101(14)100(8)99(1)98(3)93(1)*}	93(1)
$LFSR_6$	Key_{23}	{*110(1)109(1)108(2)107(13)106(28)105(17)104(15)103(13)102(14)101(7)100(7)99(3)98(2)97(2)96(1)93(1)91(1)*}	91(1)
$LFSR_6$	Key_{24}	{*109(1)108(6)107(5)106(17)105(23)104(24)103(21)102(13)101(6)100(2)99(2)98(4)97(2)96(1)95(1)*}	95(1)
$LFSR_6$	Key_{25}	{*110(1)108(6)107(11)106(11)105(17)104(22)103(16)102(16)101(8)100(3)99(3)98(9)97(4)94(1)*}	94(1)
$LFSR_6$	Key_{26}	{*109(1)108(3)107(14)106(17)105(22)104(15)103(18)102(13)101(7)100(4)99(6)98(5)97(1)96(1)94(1)*}	94(1)
$LFSR_6$	Key_{27}	{*108(8)107(15)106(20)105(13)104(12)103(19)102(15)101(10)100(4)99(4)98(2)97(4)96(1)92(1)*}	92(1)
$LFSR_6$	Key_{28}	{*109(3)108(9)107(6)106(24)105(15)104(18)103(17)102(11)101(6)100(9)99(7)98(1)94(1)92(1)*}	92(1)
$LFSR_6$	Key_{29}	{*109(3)108(6)107(9)106(14)105(18)104(16)103(16)102(10)101(15)100(10)99(4)98(3)97(2)96(1)95(1)*}	95(1)
$LFSR_7$	Key_0	{*109(1)108(9)107(14)106(16)105(23)104(15)103(17)102(13)101(9)100(7)99(2)98(1)96(1)*}	96(1)
$LFSR_7$	Key_1	{*109(1)108(2)107(9)106(16)105(25)104(24)103(20)102(6)101(7)100(6)99(5)98(3)97(1)96(3)*}	96(3)
$LFSR_7$	Key_2	{*109(2)108(5)107(9)106(21)105(25)104(17)103(19)102(13)101(9)100(3)99(4)97(1)*}	97(1)
$LFSR_7$	Key_3	{*109(2)108(3)107(6)106(22)105(19)104(23)103(14)102(16)101(10)100(6)99(3)98(1)94(3)*}	94(3)
$LFSR_7$	Key_4	{*109(2)108(7)107(9)106(19)105(16)104(25)103(18)102(8)101(9)100(6)99(5)98(1)96(2)95(1)*}	95(1)
$LFSR_7$	Key_5	{*109(2)108(5)107(9)106(16)105(19)104(15)103(20)102(16)101(6)100(8)99(7)98(1)97(3)95(1)*}	95(1)
$LFSR_7$	Key_6	{*109(1)108(5)107(9)106(15)105(29)104(25)103(15)102(10)101(7)100(4)99(5)98(1)97(2)*}	97(2)
$LFSR_7$	Key_7	{*110(1)109(1)108(4)107(16)106(19)105(15)104(19)103(18)102(13)101(10)100(5)99(3)98(1)96(1)95(1)94(1)*}	94(1)
$LFSR_7$	Key_8	{*108(5)107(7)106(18)105(22)104(22)103(18)102(11)101(7)100(7)99(6)98(3)96(2)*}	96(2)
$LFSR_7$	Key_9	{*108(5)107(9)106(16)105(17)104(23)103(12)102(18)101(10)100(3)99(4)98(1)97(5)96(2)95(2)94(1)*}	94(1)
$LFSR_7$	Key_{10}	{*109(2)108(2)107(11)106(20)105(20)104(21)103(14)102(6)101(12)100(9)99(6)98(2)96(1)95(1)93(1)*}	93(1)
$LFSR_7$	Key_{11}	{*109(3)108(3)107(8)106(20)105(17)104(23)103(18)102(7)101(13)100(8)99(3)98(2)97(2)94(1)*}	94(1)
$LFSR_7$	Key_{12}	{*109(1)108(6)107(7)106(18)105(23)104(15)103(16)102(14)101(10)100(8)99(3)98(3)97(1)96(1)95(2)*}	95(2)
$LFSR_7$	Key_{13}	{*110(1)109(2)108(3)107(10)106(25)105(18)104(15)103(17)102(13)101(8)100(9)99(1)98(2)97(3)96(1)*}	96(1)
$LFSR_7$	Key_{14}	{*110(1)109(3)108(3)107(13)106(16)105(23)104(19)103(15)102(13)101(9)100(2)99(3)98(3)96(2)94(2)89(1)*}	89(1)
$LFSR_7$	Key_{15}	{*108(3)107(7)106(8)105(33)104(19)103(20)102(10)101(11)100(6)99(4)98(5)97(2)*}	97(2)
$LFSR_7$	Key_{16}	{*109(2)108(3)107(11)106(20)105(19)104(18)103(21)102(9)101(3)100(9)99(9)97(1)96(2)93(1)*}	93(1)
$LFSR_7$	Key_{17}	{*109(1)108(4)107(10)106(17)105(19)104(24)103(16)102(12)101(6)100(4)99(7)98(3)97(3)95(1)92(1)*}	92(1)
$LFSR_7$	Key_{18}	{*108(6)107(11)106(17)105(24)104(18)103(11)102(4)101(14)100(7)99(5)98(4)97(2)96(3)93(1)91(1)*}	91(1)
$LFSR_7$	Key_{19}	{*110(1)109(5)108(2)107(11)106(12)105(19)104(22)103(13)102(15)101(10)100(5)99(1)98(4)97(1)96(3)95(2)93(1)90(1)*}	90(1)
$LFSR_7$	Key_{20}	{*108(6)107(12)106(12)105(20)104(21)103(18)102(16)101(7)100(5)99(3)98(3)97(4)95(1)*}	95(1)
$LFSR_7$	Key_{21}	{*108(5)107(13)106(23)105(27)104(12)103(16)102(13)101(7)100(7)99(1)98(1)97(2)96(1)*}	96(1)
$LFSR_7$	Key_{22}	{*109(2)108(6)107(11)106(18)105(21)104(16)103(12)102(12)101(8)100(7)99(3)98(5)97(4)96(2)95(1)*}	95(1)
$LFSR_7$	Key_{23}	{*109(1)108(5)107(14)106(15)105(20)104(16)103(19)102(11)101(10)100(9)99(3)98(2)97(1)95(2)*}	95(2)
$LFSR_7$	Key_{24}	{*109(2)108(4)107(8)106(19)105(19)104(24)103(18)102(10)101(7)100(7)99(1)98(4)97(1)96(2)95(1)94(1)*}	94(1)
$LFSR_7$	Key_{25}	{*109(2)108(5)107(3)106(14)105(18)104(15)103(23)102(17)101(12)100(6)99(5)98(3)97(3)96(2)*}	96(2)
$LFSR_7$	Key_{26}	{*109(1)108(5)107(11)106(15)105(21)104(22)103(13)102(13)101(12)100(6)99(3)98(1)97(1)96(3)94(1)*}	94(1)
$LFSR_7$	Key_{27}	{*108(3)107(14)106(14)105(22)104(16)103(20)102(15)101(9)100(4)99(7)98(3)97(1)*}	97(1)
$LFSR_7$	Key_{28}	{*109(2)108(5)107(9)106(20)105(21)104(15)103(17)102(11)101(9)100(7)99(4)98(5)97(1)96(1)91(1)*}	91(1)
$LFSR_7$	Key_{29}	{*109(2)108(8)107(9)106(16)105(21)104(18)103(16)102(8)101(13)100(8)99(5)98(3)97(1)*}	97(1)
$LFSR_8$	Key_0	{*109(2)108(2)107(13)106(18)105(17)104(17)103(16)102(18)101(9)100(9)99(4)98(1)97(1)96(1)*}	96(1)
$LFSR_8$	Key_1	{*109(4)108(2)107(11)106(16)105(21)104(20)103(14)102(10)101(10)100(6)99(3)98(3)97(5)96(1)95(1)89(1)*}	89(1)
$LFSR_8$	Key_2	{*109(2)108(4)107(12)106(18)105(16)104(22)103(22)102(11)101(10)100(3)99(4)98(2)97(2)*}	97(2)
$LFSR_8$	Key_3	{*109(2)108(4)107(6)106(16)105(25)104(19)103(19)102(8)101(6)100(11)99(2)98(3)97(2)96(2)95(1)94(2)*}	94(2)
$LFSR_8$	Key_4	{*110(1)108(4)107(15)106(10)105(18)104(24)103(20)102(11)101(9)100(2)99(6)98(3)97(1)96(2)95(1)93(1)*}	93(1)
$LFSR_8$	Key_5	{*111(1)109(1)108(5)107(7)106(20)105(16)104(21)103(22)102(17)101(8)100(2)99(1)98(2)97(4)96(1)*}	96(1)
$LFSR_8$	Key_6	{*110(1)108(3)107(14)106(22)105(19)104(19)103(15)102(12)101(10)100(5)99(3)98(3)96(1)95(1)*}	95(1)
$LFSR_8$	Key_7	{*108(3)107(4)106(22)105(15)104(19)103(12)102(14)101(17)100(4)99(7)98(5)97(3)96(2)90(1)*}	90(1)
$LFSR_8$	Key_8	{*109(1)108(5)107(14)106(19)105(22)104(20)103(18)102(9)101(8)100(6)99(4)98(2)*}	98(2)
$LFSR_8$	Key_9	{*110(1)109(1)108(4)107(9)106(10)105(20)104(29)103(20)102(9)101(8)100(5)99(6)98(2)97(1)95(2)93(1)*}	93(1)
$LFSR_8$	Key_{10}	{*109(3)108(7)107(9)106(17)105(19)104(23)103(12)102(14)101(11)100(5)99(3)98(2)97(1)96(2)*}	96(2)
$LFSR_8$	Key_{11}	{*109(1)108(8)107(13)106(14)105(21)104(21)103(14)102(11)101(6)100(9)99(6)98(1)97(2)93(1)*}	93(1)
$LFSR_8$	Key_{12}	{*110(1)108(2)107(5)106(15)105(26)104(17)103(17)102(17)101(4)100(7)99(5)98(3)97(1)96(2)95(3)94(1)93(1)92(1)*}	92(1)
$LFSR_8$	Key_{13}	{*109(2)108(8)107(12)106(13)105(17)104(15)103(22)102(11)101(13)100(5)99(1)98(5)97(1)96(1)95(2)*}	95(2)
$LFSR_8$	Key_{14}	{*109(3)108(5)107(10)106(7)105(17)104(25)103(19)102(13)101(11)100(10)99(3)98(1)97(3)94(1)*}	94(1)
$LFSR_8$	Key_{15}	{*108(4)107(16)106(8)105(23)104(16)103(21)102(14)101(9)100(4)99(8)98(1)97(2)96(1)95(1)*}	95(1)
$LFSR_8$	Key_{16}	{*110(1)109(2)108(5)107(12)106(13)105(16)104(20)103(18)102(8)101(10)100(9)99(8)98(1)97(3)96(1)94(1)*}	94(1)
$LFSR_8$	Key_{17}	{*109(1)108(3)107(16)106(12)105(20)104(14)103(20)102(7)101(14)100(5)99(9)98(3)97(2)96(1)93(1)*}	93(1)
$LFSR_8$	Key_{18}	{*109(1)108(9)107(13)106(8)105(19)104(21)103(17)102(10)101(8)100(5)99(6)98(5)97(2)96(2)95(2)*}	95(2)
$LFSR_8$	Key_{19}	{*109(3)108(4)107(12)106(17)105(22)104(15)103(14)102(18)101(10)100(5)99(3)98(1)97(1)95(1)94(1)91(1)*}	91(1)
$LFSR_8$	Key_{20}	{*110(1)108(6)107(12)106(15)105(23)104(18)103(13)102(10)101(12)100(4)99(3)98(1)97(3)96(3)95(2)94(1)92(1)*}	92(1)
$LFSR_8$	Key_{21}	{*110(2)109(1)108(3)107(13)106(22)105(21)104(16)103(17)102(9)101(6)100(6)99(3)98(5)97(2)96(2)*}	96(2)
$LFSR_8$	Key_{22}	{*109(1)108(6)107(7)106(17)105(21)104(20)103(14)102(13)101(12)100(8)99(2)98(3)97(1)95(2)94(1)*}	94(1)
$LFSR_8$	Key_{23}	{*110(1)108(3)107(8)106(18)105(18)104(18)103(14)102(17)101(6)100(10)99(6)98(3)97(3)95(1)94(1)93(1)*}	93(1)

LFSR_8	Key_{24}	{*108(3)107(12)106(17)105(34)104(15)103(19)102(8)101(6)100(6)99(1)98(2)97(1)96(1)95(1)94(1)90(1)*}	90(1)
LFSR_8	Key_{25}	{*109(1)108(4)107(13)106(17)105(15)104(17)103(19)102(17)101(7)100(7)99(6)98(2)97(2)96(1)*}	96(1)
LFSR_8	Key_{26}	{*109(5)108(8)107(13)106(14)105(22)104(21)103(10)102(9)101(14)100(2)99(2)98(2)96(3)95(2)93(1)*}	93(1)
LFSR_8	Key_{27}	{*109(1)108(2)107(12)106(22)105(21)104(21)103(15)102(11)101(7)100(5)99(6)98(3)97(2)*}	97(2)
LFSR_8	Key_{28}	{*108(10)107(10)106(20)105(12)104(18)103(16)102(12)101(11)100(10)99(2)97(4)96(2)95(1)*}	95(1)
LFSR_8	Key_{29}	{*108(7)107(10)106(19)105(15)104(23)103(11)102(12)101(15)100(2)99(6)98(2)97(3)96(1)95(1)89(1)*}	89(1)
LFSR_9	Key_{0}	{*109(1)108(3)107(12)106(12)105(26)104(21)103(21)102(9)101(10)100(5)99(3)98(3)97(1)94(1)*}	94(1)
LFSR_9	Key_{1}	{*109(1)108(4)107(6)106(25)105(24)104(18)103(17)102(9)101(6)100(11)99(4)98(2)91(1)*}	91(1)
LFSR_9	Key_{2}	{*109(1)108(1)107(8)106(20)105(21)104(20)103(19)102(16)101(10)100(4)99(1)97(4)96(1)95(1)94(1)*}	94(1)
LFSR_9	Key_{3}	{*109(2)108(7)107(10)106(19)105(24)104(15)103(11)102(13)101(9)100(5)99(4)98(4)97(1)95(2)94(2)*}	94(2)
LFSR_9	Key_{4}	{*108(6)107(14)106(12)105(24)104(18)103(16)102(11)101(11)100(8)99(4)97(1)94(1)93(2)*}	93(2)
LFSR_9	Key_{5}	{*108(6)107(9)106(16)105(16)104(28)103(14)102(21)101(9)100(4)99(2)98(1)97(2)*}	97(2)
LFSR_9	Key_{6}	{*108(3)107(13)106(15)105(19)104(20)103(17)102(3)101(11)100(10)99(5)98(2)97(1)96(6)95(3)*}	95(3)
LFSR_9	Key_{7}	{*108(2)107(12)106(16)105(27)104(18)103(15)102(13)101(8)100(6)99(3)98(2)97(2)96(2)95(1)94(1)*}	94(1)
LFSR_9	Key_{8}	{*108(3)107(7)106(20)105(29)104(13)103(15)102(19)101(7)100(6)99(7)97(1)95(1)*}	95(1)
LFSR_9	Key_{9}	{*109(2)108(4)107(15)106(18)105(12)104(21)103(11)102(21)101(9)100(8)99(2)98(1)97(3)96(1)*}	96(1)
LFSR_9	Key_{10}	{*110(1)109(2)108(2)107(15)106(16)105(16)104(19)103(17)102(10)101(14)100(6)99(4)98(4)97(2)*}	97(2)
LFSR_9	Key_{11}	{*109(3)108(10)107(14)106(18)105(14)104(21)103(12)102(10)101(7)100(7)99(6)98(3)97(2)94(1)*}	94(1)
LFSR_9	Key_{12}	{*109(1)108(5)107(14)106(19)105(21)104(22)103(15)102(13)101(5)100(6)99(5)98(1)96(1)*}	96(1)
LFSR_9	Key_{13}	{*109(1)108(5)107(13)106(19)105(21)104(18)103(8)102(11)101(11)100(7)99(7)98(4)97(1)95(2)*}	95(2)
LFSR_9	Key_{14}	{*109(3)108(3)107(9)106(15)105(15)104(13)103(23)102(18)101(12)100(6)99(5)98(2)97(3)93(1)*}	93(1)
LFSR_9	Key_{15}	{*109(1)108(5)107(13)106(14)105(19)104(18)103(17)102(10)101(9)100(4)99(9)98(2)97(3)96(3)93(1)*}	93(1)
LFSR_9	Key_{16}	{*108(7)107(12)106(11)105(21)104(14)103(17)102(24)101(8)100(6)99(3)98(1)97(3)95(1)*}	95(1)
LFSR_9	Key_{17}	{*110(1)109(2)108(4)107(10)106(22)105(19)104(14)103(12)102(11)101(13)100(6)99(3)98(2)97(5)96(3)95(1)*}	95(1)
LFSR_9	Key_{18}	{*109(3)108(3)107(12)106(22)105(13)104(22)103(18)102(6)101(11)100(5)99(5)98(2)97(2)96(3)94(1)*}	94(1)
LFSR_9	Key_{19}	{*108(4)107(10)106(23)105(18)104(21)103(15)102(8)101(11)100(5)99(4)98(5)96(2)95(1)94(1)*}	94(1)
LFSR_9	Key_{20}	{*109(2)108(5)107(7)106(21)105(22)104(14)103(16)102(18)101(11)100(2)99(6)98(1)97(1)96(2)*}	96(2)
LFSR_9	Key_{21}	{*108(9)107(7)106(19)105(25)104(14)103(16)102(9)101(13)100(6)99(4)98(3)97(2)95(1)*}	95(1)
LFSR_9	Key_{22}	{*109(3)108(3)107(16)106(15)105(15)104(24)103(12)102(15)101(10)100(4)99(9)97(1)96(1)*}	96(1)
LFSR_9	Key_{23}	{*109(2)108(4)107(18)106(19)105(22)104(21)103(11)102(8)101(8)100(9)99(3)98(1)97(1)95(1)*}	95(1)
LFSR_9	Key_{24}	{*110(1)109(1)108(5)107(17)106(17)105(15)104(21)103(18)102(16)101(6)100(4)99(1)98(2)97(1)96(1)95(2)*}	95(2)
LFSR_9	Key_{25}	{*109(1)108(5)107(7)106(24)105(21)104(19)103(18)102(9)101(11)100(6)99(4)97(2)89(1)*}	89(1)
LFSR_9	Key_{26}	{*108(4)107(11)106(11)105(24)104(14)103(16)102(16)101(11)100(5)99(4)98(1)97(3)96(5)95(1)94(1)91(1)*}	91(1)
LFSR_9	Key_{27}	{*109(2)108(3)107(11)106(19)105(24)104(15)103(12)102(12)101(12)100(9)99(6)98(2)96(1)*}	96(1)
LFSR_9	Key_{28}	{*109(2)108(3)107(13)106(16)105(25)104(13)103(18)102(12)101(9)100(8)99(4)98(2)97(1)96(1)94(1)*}	94(1)
LFSR_9	Key_{29}	{*108(5)107(11)106(22)105(22)104(14)103(19)102(12)101(5)100(11)99(2)98(1)97(3)94(1)*}	94(1)
LFSR_{10}	Key_{0}	{*109(1)108(3)107(5)106(18)105(19)104(24)103(19)102(10)101(9)100(8)99(3)98(5)97(3)94(1)*}	94(1)
LFSR_{10}	Key_{1}	{*109(1)108(7)107(8)106(20)105(18)104(15)103(19)102(16)101(7)100(3)99(6)97(3)96(1)95(1)94(1)92(2)*}	92(2)
LFSR_{10}	Key_{2}	{*109(2)108(7)107(7)106(16)105(23)104(23)103(14)102(15)101(5)100(7)99(3)98(3)97(1)96(1)95(1)*}	95(1)
LFSR_{10}	Key_{3}	{*109(1)108(7)107(8)106(16)105(26)104(15)103(16)102(12)101(7)100(7)99(2)98(5)97(2)96(2)95(1)90(1)*}	90(1)
LFSR_{10}	Key_{4}	{*109(1)108(11)107(3)106(16)105(16)104(23)103(17)102(16)101(10)100(7)99(2)98(2)97(4)*}	97(4)
LFSR_{10}	Key_{5}	{*109(2)108(10)107(11)106(16)105(14)104(22)103(15)102(10)101(8)100(9)99(5)98(4)96(1)93(1)*}	93(1)
LFSR_{10}	Key_{6}	{*110(2)109(2)108(3)107(8)106(16)105(14)104(28)103(19)102(10)101(6)100(5)99(6)98(5)97(2)96(2)*}	96(2)
LFSR_{10}	Key_{7}	{*110(1)108(4)107(7)106(19)105(12)104(23)103(22)102(8)101(16)100(6)99(2)98(1)97(3)96(1)94(1)93(1)91(1)*}	91(1)
LFSR_{10}	Key_{8}	{*110(1)108(8)107(12)106(13)105(27)104(21)103(12)102(7)101(12)100(7)99(2)98(3)97(2)96(1)*}	96(1)
LFSR_{10}	Key_{9}	{*109(2)108(5)107(10)106(20)105(18)104(17)103(15)102(12)101(10)100(1)99(7)98(7)97(3)95(1)*}	95(1)
LFSR_{10}	Key_{10}	{*109(2)108(2)107(7)106(16)105(18)104(11)103(14)102(20)101(9)100(9)99(8)98(6)97(3)96(3)*}	96(3)
LFSR_{10}	Key_{11}	{*110(1)109(2)108(4)107(10)106(15)105(22)104(17)103(21)102(17)101(8)100(3)99(2)98(2)97(2)94(1)92(1)*}	92(1)
LFSR_{10}	Key_{12}	{*109(2)108(4)107(12)106(20)105(22)104(16)103(18)102(8)101(10)100(4)99(5)98(3)97(2)96(2)*}	96(2)
LFSR_{10}	Key_{13}	{*109(1)108(3)107(9)106(15)105(21)104(19)103(22)102(12)101(9)100(4)99(4)98(3)97(1)96(3)95(2)*}	95(2)
LFSR_{10}	Key_{14}	{*109(1)108(2)107(11)106(19)105(18)104(29)103(6)102(13)101(1)100(9)99(10)98(4)97(1)96(1)95(2)94(1)*}	94(1)
LFSR_{10}	Key_{15}	{*110(1)109(2)108(6)107(3)106(12)105(21)104(22)103(17)102(20)101(12)100(5)99(2)98(2)97(2)94(1)*}	94(1)
LFSR_{10}	Key_{16}	{*109(3)108(4)107(10)106(24)105(14)104(21)103(18)102(12)101(10)100(5)99(2)98(1)97(3)94(1)*}	94(1)
LFSR_{10}	Key_{17}	{*109(2)108(4)107(7)106(14)105(30)104(18)103(19)102(11)101(7)100(11)99(3)97(1)96(1)*}	96(1)
LFSR_{10}	Key_{18}	{*110(1)109(2)108(4)107(9)106(18)105(14)104(28)103(19)102(11)101(8)100(7)99(4)98(1)97(1)96(1)*}	96(1)
LFSR_{10}	Key_{19}	{*109(1)108(5)107(9)106(19)105(17)104(12)103(22)102(16)101(13)100(7)99(2)98(3)97(1)96(1)*}	96(1)
LFSR_{10}	Key_{20}	{*109(3)108(5)107(13)106(17)105(23)104(12)103(15)102(12)101(10)100(6)99(3)98(3)97(3)96(1)95(2)*}	95(2)
LFSR_{10}	Key_{21}	{*108(4)107(13)106(12)105(24)104(17)103(15)102(13)101(7)100(6)99(5)98(6)97(3)96(1)95(1)91(1)*}	91(1)
LFSR_{10}	Key_{22}	{*109(1)108(8)107(12)106(19)105(12)104(26)103(11)102(11)101(7)100(5)99(8)98(2)97(4)96(1)95(1)*}	95(1)
LFSR_{10}	Key_{23}	{*109(2)108(2)107(13)106(11)105(19)104(27)103(16)102(8)101(11)100(6)99(8)98(3)97(2)*}	97(2)
LFSR_{10}	Key_{24}	{*108(5)107(11)106(15)105(18)104(20)103(14)102(16)101(11)100(6)99(3)98(2)97(3)96(3)95(1)*}	95(1)
LFSR_{10}	Key_{25}	{*109(1)108(5)107(11)106(21)105(17)104(23)103(14)102(11)101(9)100(7)99(3)98(1)97(1)96(3)95(1)*}	95(1)
LFSR_{10}	Key_{26}	{*110(1)109(2)108(6)107(8)106(19)105(23)104(15)103(22)102(11)101(6)100(8)99(4)98(1)96(2)*}	96(2)

$LFSR_{10}$	Key_{27}	{∗109(1)108(2)107(8)106(15)105(23)104(21)103(11)102(13)101(11)100(6)99(8)98(5)97(2)96(2)∗}	96(2)
$LFSR_{10}$	Key_{28}	{∗109(5)108(9)107(9)106(20)105(12)104(19)103(19)102(16)101(9)100(4)99(4)98(2)∗}	98(2)
$LFSR_{10}$	Key_{29}	{∗109(2)108(4)107(13)106(10)105(18)104(24)103(14)102(9)101(10)100(15)99(3)97(3)95(1)94(1)93(1)∗}	93(1)
$LFSR_{11}$	Key_{0}	{∗109(1)108(6)107(8)106(17)105(15)104(15)103(23)102(7)101(17)100(9)99(4)98(2)97(1)96(1)95(1)90(1)∗}	90(1)
$LFSR_{11}$	Key_{1}	{∗109(1)108(6)107(6)106(14)105(22)104(23)103(16)102(18)101(9)100(4)99(2)98(1)97(1)96(2)95(2)93(1)∗}	93(1)
$LFSR_{11}$	Key_{2}	{∗110(3)109(3)108(7)107(10)106(13)105(25)104(20)103(14)102(9)101(9)100(3)99(3)98(4)97(2)95(1)94(1)93(1)∗}	93(1)
$LFSR_{11}$	Key_{3}	{∗108(2)107(10)106(15)105(21)104(23)103(19)102(13)101(8)100(7)99(4)98(1)97(3)96(1)95(1)∗}	95(1)
$LFSR_{11}$	Key_{4}	{∗109(6)108(2)107(11)106(19)105(17)104(14)103(20)102(13)101(8)100(7)99(7)96(1)95(1)94(1)93(1)∗}	93(1)
$LFSR_{11}$	Key_{5}	{∗109(2)108(1)107(9)106(16)105(16)104(22)103(17)102(20)101(8)100(3)99(4)98(4)97(3)96(3)∗}	96(3)
$LFSR_{11}$	Key_{6}	{∗109(2)108(4)107(12)106(12)105(20)104(13)103(20)102(8)101(11)100(9)99(4)98(4)97(2)96(2)95(1)93(3)92(1)∗}	92(1)
$LFSR_{11}$	Key_{7}	{∗109(1)108(4)107(7)106(18)105(21)104(19)103(18)102(12)101(12)100(8)99(4)98(1)96(2)94(1)∗}	94(1)
$LFSR_{11}$	Key_{8}	{∗108(8)107(12)106(18)105(17)104(20)103(11)102(12)101(9)100(4)99(7)98(1)97(4)96(2)95(1)94(1)93(1)∗}	93(1)
$LFSR_{11}$	Key_{9}	{∗109(1)108(6)107(14)106(18)105(17)104(20)103(18)102(8)101(9)100(6)99(4)98(3)97(2)96(2)∗}	96(2)
$LFSR_{11}$	Key_{10}	{∗109(2)108(4)107(6)106(20)105(27)104(19)103(13)102(7)101(12)100(7)99(4)98(2)97(3)96(2)∗}	96(2)
$LFSR_{11}$	Key_{11}	{∗109(2)108(6)107(10)106(11)105(20)104(25)103(12)102(14)101(8)100(10)99(4)98(2)95(1)94(1)93(1)90(1)∗}	90(1)
$LFSR_{11}$	Key_{12}	{∗109(2)108(2)107(9)106(13)105(24)104(18)103(15)102(19)101(10)100(6)99(4)98(3)97(2)95(1)∗}	95(1)
$LFSR_{11}$	Key_{13}	{∗110(1)108(2)107(9)106(17)105(17)104(15)103(14)102(17)101(13)100(12)99(8)98(2)96(1)∗}	96(1)
$LFSR_{11}$	Key_{14}	{∗109(1)108(4)107(11)106(18)105(17)104(28)103(11)102(9)101(10)100(8)99(6)98(1)97(1)96(2)95(1)∗}	95(1)
$LFSR_{11}$	Key_{15}	{∗108(8)107(8)106(8)105(23)104(18)103(11)102(17)101(17)100(7)99(5)98(2)97(1)96(1)95(2)∗}	95(2)
$LFSR_{11}$	Key_{16}	{∗109(1)108(6)107(10)106(11)105(23)104(23)103(12)102(11)101(9)100(6)99(8)98(3)97(1)96(1)95(1)93(1)92(1)∗}	92(1)
$LFSR_{11}$	Key_{17}	{∗109(1)108(5)107(11)106(15)105(22)104(16)103(19)102(13)101(12)100(6)99(2)98(3)96(1)95(1)94(1)∗}	94(1)
$LFSR_{11}$	Key_{18}	{∗109(2)108(5)107(10)106(14)105(16)104(19)103(12)102(16)101(14)100(10)99(3)98(2)97(2)96(1)93(1)92(1)∗}	92(1)
$LFSR_{11}$	Key_{19}	{∗110(1)109(2)108(8)107(8)106(18)105(16)104(20)103(13)102(13)101(15)100(6)99(3)98(2)97(3)∗}	97(3)
$LFSR_{11}$	Key_{20}	{∗109(2)108(2)107(7)106(14)105(19)104(18)103(23)102(16)101(12)100(6)99(7)98(1)94(1)∗}	94(1)
$LFSR_{11}$	Key_{21}	{∗108(6)107(8)106(18)105(18)104(17)103(15)102(10)101(8)100(13)99(11)98(1)97(2)95(1)∗}	95(1)
$LFSR_{11}$	Key_{22}	{∗110(1)109(1)108(4)107(12)106(18)105(19)104(27)103(14)102(7)101(7)100(6)99(4)98(4)97(3)96(1)∗}	96(1)
$LFSR_{11}$	Key_{23}	{∗110(2)108(5)107(12)106(16)105(21)104(23)103(16)102(8)101(12)100(2)99(4)98(3)97(2)95(1)94(1)∗}	94(1)
$LFSR_{11}$	Key_{24}	{∗109(1)108(6)107(11)106(14)105(21)104(21)103(14)102(15)101(9)100(6)99(3)98(5)97(1)95(1)∗}	95(1)
$LFSR_{11}$	Key_{25}	{∗109(1)108(6)107(8)106(15)105(28)104(23)103(11)102(12)101(9)100(8)99(4)98(2)96(1)∗}	96(1)
$LFSR_{11}$	Key_{26}	{∗109(2)108(5)107(9)106(19)105(15)104(19)103(21)102(8)101(12)100(8)99(5)98(1)97(2)96(1)94(1)∗}	94(1)
$LFSR_{11}$	Key_{27}	{∗109(1)108(6)107(7)106(13)105(19)104(21)103(21)102(15)101(10)100(4)99(4)98(2)97(1)96(4)∗}	96(4)
$LFSR_{11}$	Key_{28}	{∗109(2)108(4)107(15)106(22)105(14)104(18)103(22)102(12)101(5)100(5)99(2)98(5)97(2)∗}	97(2)
$LFSR_{11}$	Key_{29}	{∗110(1)109(1)108(7)107(14)106(14)105(19)104(17)103(23)102(13)101(6)100(6)99(1)98(2)97(1)96(2)94(1)∗}	94(1)
$LFSR_{12}$	Key_{0}	{∗108(3)107(9)106(20)105(21)104(20)103(16)102(13)101(9)100(4)99(6)98(3)97(2)96(1)95(1)∗}	95(1)
$LFSR_{12}$	Key_{1}	{∗111(1)109(1)108(6)107(12)106(16)105(18)104(18)103(13)102(15)101(9)100(6)99(4)98(5)97(3)91(1)∗}	91(1)
$LFSR_{12}$	Key_{2}	{∗110(1)109(3)108(7)107(12)106(13)105(15)104(17)103(12)102(14)101(12)100(10)99(3)98(7)97(1)96(1)∗}	96(1)
$LFSR_{12}$	Key_{3}	{∗109(2)108(4)107(11)106(18)105(19)104(18)103(19)102(17)101(10)100(7)99(1)98(1)97(1)∗}	97(1)
$LFSR_{12}$	Key_{4}	{∗108(3)107(11)106(19)105(21)104(15)103(24)102(19)101(5)100(4)99(2)98(2)97(1)96(1)93(1)∗}	93(1)
$LFSR_{12}$	Key_{5}	{∗109(2)108(4)107(17)106(13)105(25)104(22)103(14)102(8)101(5)100(8)99(3)98(6)95(1)∗}	95(1)
$LFSR_{12}$	Key_{6}	{∗109(2)108(3)107(12)106(14)105(23)104(14)103(21)102(11)101(9)100(7)99(5)98(4)97(2)94(1)∗}	94(1)
$LFSR_{12}$	Key_{7}	{∗111(1)110(1)108(4)107(9)106(20)105(19)104(18)103(14)102(16)101(9)100(6)99(4)97(1)96(2)95(2)94(1)93(1)∗}	93(1)
$LFSR_{12}$	Key_{8}	{∗110(2)108(6)107(13)106(13)105(22)104(17)103(20)102(13)101(7)100(2)99(9)98(1)97(2)95(1)∗}	95(1)
$LFSR_{12}$	Key_{9}	{∗108(4)107(5)106(24)105(22)104(20)103(20)102(11)101(5)100(4)99(6)98(2)97(2)96(2)94(1)∗}	94(1)
$LFSR_{12}$	Key_{10}	{∗109(3)108(2)107(14)106(17)105(20)104(17)103(14)102(13)101(9)100(6)99(7)98(2)97(1)96(1)95(2)∗}	95(2)
$LFSR_{12}$	Key_{11}	{∗109(2)108(3)107(17)106(20)105(17)104(18)103(9)102(14)101(7)100(9)99(3)98(5)97(1)96(1)94(2)∗}	94(2)
$LFSR_{12}$	Key_{12}	{∗109(1)108(6)107(8)106(23)105(16)104(20)103(19)102(9)101(8)100(8)99(4)98(3)97(1)96(1)94(1)∗}	94(1)
$LFSR_{12}$	Key_{13}	{∗109(1)108(5)107(8)106(16)105(24)104(19)103(11)102(12)101(8)100(11)99(7)98(2)97(1)96(1)94(1)93(1)∗}	93(1)
$LFSR_{12}$	Key_{14}	{∗109(2)108(3)107(7)106(18)105(20)104(20)103(18)102(13)101(8)100(9)99(5)98(1)97(3)96(1)∗}	96(1)
$LFSR_{12}$	Key_{15}	{∗109(2)108(6)107(11)106(15)105(26)104(13)103(18)102(11)101(11)100(9)99(1)98(2)97(2)96(1)∗}	96(1)
$LFSR_{12}$	Key_{16}	{∗109(1)108(5)107(11)106(17)105(23)104(18)103(14)102(11)101(6)100(7)99(3)98(6)97(3)96(2)91(1)∗}	91(1)
$LFSR_{12}$	Key_{17}	{∗109(4)108(2)107(13)106(15)105(19)104(21)103(14)102(10)101(7)100(7)99(6)98(3)97(2)96(1)95(1)94(2)93(1)∗}	93(1)
$LFSR_{12}$	Key_{18}	{∗108(7)107(10)106(13)105(19)104(18)103(25)102(8)101(12)100(6)99(2)98(2)97(2)95(3)94(1)∗}	94(1)
$LFSR_{12}$	Key_{19}	{∗109(1)108(7)107(12)106(13)105(17)104(14)103(21)102(15)101(13)100(5)99(4)98(2)97(3)93(1)∗}	93(1)
$LFSR_{12}$	Key_{20}	{∗109(1)108(9)107(8)106(16)105(13)104(26)103(15)102(9)101(8)100(10)99(4)98(2)97(3)96(1)95(2)94(1)∗}	94(1)
$LFSR_{12}$	Key_{21}	{∗108(5)107(7)106(11)105(24)104(25)103(16)102(12)101(9)100(8)99(3)98(5)97(2)96(1)∗}	96(1)
$LFSR_{12}$	Key_{22}	{∗109(2)108(3)107(14)106(19)105(15)104(15)103(25)102(13)101(5)100(10)99(3)97(2)95(2)∗}	95(2)
$LFSR_{12}$	Key_{23}	{∗109(1)108(7)107(13)106(11)105(25)104(21)103(10)102(13)101(7)100(7)99(6)98(3)97(1)96(2)94(1)∗}	94(1)
$LFSR_{12}$	Key_{24}	{∗109(2)108(4)107(9)106(19)105(24)104(14)103(14)102(16)101(11)100(6)99(3)98(3)97(2)96(1)∗}	96(1)
$LFSR_{12}$	Key_{25}	{∗110(1)109(2)108(4)107(10)106(13)105(14)104(26)103(17)102(15)101(7)100(7)99(5)98(5)96(1)94(1)∗}	94(1)
$LFSR_{12}$	Key_{26}	{∗109(2)108(3)107(8)106(12)105(15)104(16)103(23)102(17)101(14)100(9)99(3)98(1)97(2)96(3)∗}	96(3)
$LFSR_{12}$	Key_{27}	{∗110(1)109(3)108(2)107(10)106(21)105(19)104(17)103(24)102(11)101(6)100(5)99(5)98(2)97(2)∗}	97(2)
$LFSR_{12}$	Key_{28}	{∗108(11)107(14)106(13)105(15)104(27)103(16)102(8)101(6)100(5)99(7)98(3)97(1)96(1)95(1)∗}	95(1)
$LFSR_{12}$	Key_{29}	{∗109(1)108(6)107(9)106(17)105(14)104(19)103(23)102(14)101(5)100(8)99(6)98(2)97(1)95(1)94(1)93(1)∗}	93(1)

LFSR_{13}	Key_{0}	{∗110(1)109(1)108(4)107(7)106(12)105(15)104(19)103(22)102(18)101(8)100(7)99(7)98(5)95(1)94(1)∗}	94(1)
LFSR_{13}	Key_{1}	{∗110(1)108(4)107(9)106(16)105(20)104(22)103(16)102(17)101(5)100(6)99(5)98(2)97(3)95(1)94(1)∗}	94(1)
LFSR_{13}	Key_{2}	{∗109(1)108(9)107(14)106(13)105(13)104(23)103(9)102(12)101(7)100(9)99(6)98(5)97(4)96(1)94(2)∗}	94(2)
LFSR_{13}	Key_{3}	{∗109(3)108(6)107(10)106(12)105(15)104(21)103(23)102(11)101(7)100(5)99(6)98(4)97(4)94(1)∗}	94(1)
LFSR_{13}	Key_{4}	{∗109(3)108(4)107(9)106(12)105(26)104(23)103(18)102(10)101(7)100(5)99(4)98(4)97(1)96(2)∗}	96(2)
LFSR_{13}	Key_{5}	{∗109(1)108(4)107(11)106(8)105(22)104(28)103(14)102(11)101(12)100(6)99(4)98(6)92(1)∗}	92(1)
LFSR_{13}	Key_{6}	{∗109(2)108(6)107(7)106(26)105(15)104(24)103(9)102(9)101(15)100(5)99(2)98(2)97(3)96(2)95(1)∗}	95(1)
LFSR_{13}	Key_{7}	{∗109(1)108(6)107(9)106(9)105(29)104(14)103(25)102(9)101(7)100(5)99(4)98(6)97(2)96(2)∗}	96(2)
LFSR_{13}	Key_{8}	{∗109(2)108(3)107(5)106(15)105(20)104(29)103(17)102(17)101(5)100(7)99(5)98(2)96(1)∗}	96(1)
LFSR_{13}	Key_{9}	{∗109(2)108(4)107(14)106(15)105(13)104(23)103(13)102(18)101(9)100(8)99(1)98(3)97(3)96(1)93(1)∗}	93(1)
LFSR_{13}	Key_{10}	{∗108(2)107(8)106(10)105(28)104(23)103(13)102(15)101(12)100(5)99(6)98(2)97(1)96(1)94(1)92(1)∗}	92(1)
LFSR_{13}	Key_{11}	{∗109(3)108(3)107(11)106(16)105(11)104(19)103(13)102(21)101(12)100(5)99(5)98(2)97(4)96(3)∗}	96(3)
LFSR_{13}	Key_{12}	{∗109(2)108(4)107(7)106(25)105(13)104(19)103(25)102(12)101(6)100(5)99(3)98(3)97(2)95(1)94(1)∗}	94(1)
LFSR_{13}	Key_{13}	{∗109(3)108(8)107(14)106(16)105(14)104(13)103(17)102(17)101(10)100(6)99(5)98(2)97(2)95(1)∗}	95(1)
LFSR_{13}	Key_{14}	{∗109(2)108(1)107(8)106(21)105(24)104(18)103(12)102(14)101(14)100(8)99(2)98(2)97(1)95(1)∗}	95(1)
LFSR_{13}	Key_{15}	{∗109(2)108(1)107(8)106(15)105(29)104(23)103(14)102(8)101(6)100(12)99(1)98(4)97(2)96(2)94(1)∗}	94(1)
LFSR_{13}	Key_{16}	{∗109(5)108(2)107(8)106(10)105(16)104(23)103(23)102(11)101(11)100(4)99(9)98(2)97(3)92(1)∗}	92(1)
LFSR_{13}	Key_{17}	{∗110(2)109(1)108(5)107(13)106(16)105(16)104(19)103(17)102(14)101(8)100(6)99(1)98(2)97(5)96(1)94(1)92(1)∗}	92(1)
LFSR_{13}	Key_{18}	{∗109(1)108(4)107(14)106(9)105(18)104(24)103(19)102(15)101(13)100(5)99(3)98(1)95(2)∗}	95(2)
LFSR_{13}	Key_{19}	{∗110(1)109(2)108(2)107(15)106(5)105(22)104(12)103(15)102(18)101(12)100(11)99(7)98(3)97(1)96(2)∗}	96(2)
LFSR_{13}	Key_{20}	{∗110(1)108(5)107(14)106(15)105(19)104(15)103(16)102(13)101(11)100(9)99(4)97(3)96(1)94(1)92(1)∗}	92(1)
LFSR_{13}	Key_{21}	{∗109(1)108(5)107(13)106(17)105(13)104(13)103(18)102(13)101(11)100(11)99(7)98(3)97(3)∗}	97(3)
LFSR_{13}	Key_{22}	{∗109(4)108(3)107(13)106(14)105(20)104(16)103(24)102(9)101(7)100(4)99(5)98(1)97(4)96(2)95(1)91(1)∗}	91(1)
LFSR_{13}	Key_{23}	{∗111(1)108(5)107(13)106(22)105(20)104(23)103(14)102(7)101(4)100(2)99(5)98(7)96(2)94(1)93(2)∗}	93(2)
LFSR_{13}	Key_{24}	{∗109(1)108(3)107(8)106(18)105(20)104(21)103(22)102(11)101(12)100(7)99(2)98(3)∗}	98(3)
LFSR_{13}	Key_{25}	{∗110(1)108(5)107(12)106(20)105(20)104(21)103(17)102(8)101(10)100(7)99(3)98(2)97(2)∗}	97(2)
LFSR_{13}	Key_{26}	{∗109(2)108(10)107(6)106(16)105(17)104(16)103(15)102(8)101(14)100(6)99(9)98(3)97(3)96(1)95(1)92(1)∗}	92(1)
LFSR_{13}	Key_{27}	{∗109(1)108(2)107(15)106(19)105(16)104(19)103(15)102(10)101(9)100(12)99(2)98(2)97(3)96(3)∗}	96(3)
LFSR_{13}	Key_{28}	{∗108(2)107(16)106(20)105(17)104(18)103(22)102(8)101(12)100(4)99(1)98(3)97(1)96(1)95(3)∗}	95(3)
LFSR_{13}	Key_{29}	{∗108(2)107(14)106(16)105(16)104(21)103(16)102(11)101(11)100(5)99(7)98(4)97(1)96(1)95(1)93(1)90(1)∗}	90(1)
LFSR_{14}	Key_{0}	{∗110(1)109(3)108(3)107(9)106(25)105(15)104(14)103(18)102(11)101(10)100(5)99(4)98(5)97(1)96(1)95(1)94(1)93(1)∗}	93(1)
LFSR_{14}	Key_{1}	{∗109(1)108(8)107(8)106(16)105(22)104(18)103(19)102(11)101(6)100(9)99(3)98(1)97(5)96(1)∗}	96(1)
LFSR_{14}	Key_{2}	{∗109(2)108(3)107(19)106(14)105(16)104(19)103(10)102(10)101(12)100(10)99(6)98(4)96(2)93(1)∗}	93(1)
LFSR_{14}	Key_{3}	{∗108(6)107(12)106(16)105(17)104(17)103(18)102(8)101(13)100(7)99(6)98(3)97(1)96(2)95(1)91(1)∗}	91(1)
LFSR_{14}	Key_{4}	{∗109(3)108(3)107(14)106(17)105(20)104(17)103(17)102(16)101(4)100(8)99(5)97(2)95(2)∗}	95(2)
LFSR_{14}	Key_{5}	{∗110(1)108(3)107(7)106(23)105(16)104(19)103(20)102(7)101(12)100(8)99(3)98(4)97(3)96(1)94(1)∗}	94(1)
LFSR_{14}	Key_{6}	{∗108(3)107(8)106(21)105(14)104(14)103(16)102(17)101(13)100(11)99(4)98(1)97(2)96(1)94(1)92(1)90(1)∗}	90(1)
LFSR_{14}	Key_{7}	{∗110(1)109(1)108(12)107(8)106(12)105(21)104(22)103(11)102(13)101(6)100(9)99(7)98(2)96(3)∗}	96(3)
LFSR_{14}	Key_{8}	{∗109(2)108(4)107(11)106(15)105(13)104(22)103(14)102(18)101(7)100(8)99(5)98(2)97(1)96(3)95(3)∗}	95(3)
LFSR_{14}	Key_{9}	{∗108(5)107(6)106(18)105(15)104(19)103(20)102(13)101(10)100(8)99(7)98(3)97(1)95(1)94(2)∗}	94(2)
LFSR_{14}	Key_{10}	{∗108(4)107(17)106(17)105(13)104(20)103(16)102(11)101(14)100(2)99(4)98(4)97(2)96(2)95(1)94(1)∗}	94(1)
LFSR_{14}	Key_{11}	{∗109(1)108(6)107(11)106(15)105(20)104(24)103(13)102(14)101(7)100(6)99(4)98(4)97(2)93(1)∗}	93(1)
LFSR_{14}	Key_{12}	{∗110(1)109(1)108(4)107(14)106(24)105(17)104(17)103(18)102(13)101(9)100(4)99(1)98(2)96(1)95(2)∗}	95(2)
LFSR_{14}	Key_{13}	{∗109(4)108(4)107(10)106(16)105(22)104(17)103(14)102(9)101(11)100(12)99(4)98(1)97(4)∗}	97(4)
LFSR_{14}	Key_{14}	{∗109(2)108(5)107(10)106(14)105(26)104(13)103(11)102(16)101(9)100(8)99(8)98(3)95(2)90(1)∗}	90(1)
LFSR_{14}	Key_{15}	{∗109(3)108(4)107(8)106(15)105(23)104(22)103(22)102(6)101(11)100(7)99(2)98(3)97(1)95(1)∗}	95(1)
LFSR_{14}	Key_{16}	{∗109(1)108(5)107(10)106(12)105(24)104(16)103(16)102(13)101(11)100(4)99(5)98(4)97(1)96(4)92(1)90(1)∗}	90(1)
LFSR_{14}	Key_{17}	{∗109(3)108(3)107(7)106(20)105(18)104(24)103(14)102(20)101(7)100(4)99(2)98(4)97(1)95(1)∗}	95(1)
LFSR_{14}	Key_{18}	{∗109(1)108(3)107(19)106(21)105(20)104(20)103(12)102(10)101(8)100(6)99(2)98(2)97(4)∗}	97(4)
LFSR_{14}	Key_{19}	{∗109(1)108(2)107(12)106(19)105(19)104(14)103(20)102(9)101(10)100(8)99(9)98(3)97(1)96(1)∗}	96(1)
LFSR_{14}	Key_{20}	{∗108(6)107(11)106(18)105(24)104(19)103(15)102(12)101(10)100(2)99(4)98(4)96(1)94(1)92(1)∗}	92(1)
LFSR_{14}	Key_{21}	{∗108(3)107(11)106(16)105(21)104(19)103(15)102(12)101(11)100(5)99(4)98(5)97(2)96(3)95(1)∗}	95(1)
LFSR_{14}	Key_{22}	{∗109(2)108(5)107(13)106(11)105(18)104(17)103(25)102(12)101(9)100(3)99(6)98(3)97(2)96(2)∗}	96(2)
LFSR_{14}	Key_{23}	{∗109(3)108(6)107(9)106(13)105(16)104(18)103(12)102(14)101(19)100(6)99(5)98(3)97(2)96(1)92(1)∗}	92(1)
LFSR_{14}	Key_{24}	{∗109(1)108(4)107(13)106(14)105(17)104(20)103(12)102(14)101(8)100(6)99(6)98(5)97(5)95(1)92(2)∗}	92(2)
LFSR_{14}	Key_{25}	{∗108(12)107(10)106(17)105(25)104(15)103(15)102(6)101(11)100(6)99(6)98(4)97(1)∗}	97(1)
LFSR_{14}	Key_{26}	{∗108(6)107(9)106(10)105(27)104(27)103(18)102(7)101(10)100(4)99(4)97(3)95(1)94(1)90(1)∗}	90(1)
LFSR_{14}	Key_{27}	{∗110(1)109(2)108(3)107(7)106(14)105(15)104(18)103(17)102(19)101(15)100(6)99(4)98(2)97(2)96(1)95(2)∗}	95(2)
LFSR_{14}	Key_{28}	{∗109(2)108(5)107(14)106(12)105(29)104(14)103(17)102(16)101(5)100(4)99(5)98(3)96(1)95(1)∗}	95(1)
LFSR_{14}	Key_{29}	{∗109(2)108(5)107(14)106(14)105(25)104(17)103(19)102(11)101(5)100(9)99(3)98(1)97(2)96(1)∗}	96(1)
LFSR_{15}	Key_{0}	{∗109(2)108(4)107(11)106(12)105(23)104(11)103(14)102(20)101(12)100(9)99(2)98(2)97(2)96(1)95(2)93(1)∗}	93(1)
LFSR_{15}	Key_{1}	{∗110(2)109(2)108(3)107(12)106(15)105(15)104(17)103(14)102(13)101(8)100(14)99(6)98(3)97(1)95(1)94(1)89(1)∗}	89(1)
LFSR_{15}	Key_{2}	{∗108(5)107(9)106(14)105(23)104(28)103(20)102(7)101(5)100(8)99(5)98(1)97(1)96(1)93(1)∗}	93(1)

LFSR$_{15}$	Key_3	{∗110(2)109(2)108(4)107(7)106(19)105(22)104(20)103(15)102(14)101(4)100(8)99(3)98(4)96(2)92(1)91(1)∗}	91(1)
LFSR$_{15}$	Key_4	{∗108(7)107(12)106(8)105(22)104(14)103(15)102(15)101(10)100(4)99(7)98(8)97(2)96(1)95(2)94(1)∗}	94(1)
LFSR$_{15}$	Key_5	{∗108(4)107(12)106(16)105(13)104(23)103(17)102(11)101(11)100(8)99(8)98(1)97(3)95(1)∗}	95(1)
LFSR$_{15}$	Key_6	{∗109(2)108(3)107(5)106(14)105(20)104(18)103(17)102(19)101(12)100(7)99(2)98(8)94(1)∗}	94(1)
LFSR$_{15}$	Key_7	{∗109(5)108(5)107(5)106(17)105(18)104(22)103(19)102(13)101(8)100(2)99(6)98(3)97(2)96(1)94(1)90(1)∗}	90(1)
LFSR$_{15}$	Key_8	{∗110(1)108(4)107(16)106(14)105(16)104(18)103(20)102(11)101(8)100(4)99(5)98(2)97(4)96(4)92(1)∗}	92(1)
LFSR$_{15}$	Key_9	{∗109(2)108(4)107(5)106(21)105(20)104(26)103(15)102(12)101(5)100(5)99(4)98(4)97(1)96(3)93(1)∗}	93(1)
LFSR$_{15}$	Key_{10}	{∗109(1)108(10)107(14)106(10)105(16)104(20)103(17)102(13)101(11)100(7)99(3)98(2)97(2)96(1)94(1)∗}	94(1)
LFSR$_{15}$	Key_{11}	{∗109(1)108(6)107(11)106(18)105(18)104(20)103(17)102(16)101(6)100(7)99(3)98(4)94(1)∗}	94(1)
LFSR$_{15}$	Key_{12}	{∗109(1)108(5)107(10)106(15)105(22)104(25)103(15)102(9)101(9)100(3)99(4)98(3)97(4)96(1)95(1)94(1)∗}	94(1)
LFSR$_{15}$	Key_{13}	{∗109(1)108(5)107(3)106(13)105(19)104(29)103(16)102(15)101(13)100(11)98(2)96(1)∗}	96(1)
LFSR$_{15}$	Key_{14}	{∗109(2)108(2)107(11)106(21)105(20)104(18)103(18)102(8)101(9)100(6)99(4)98(3)97(1)96(1)95(2)94(2)∗}	94(2)
LFSR$_{15}$	Key_{15}	{∗109(2)108(5)107(7)106(13)105(20)104(24)103(23)102(6)101(14)100(5)99(2)98(4)97(2)95(1)∗}	95(1)
LFSR$_{15}$	Key_{16}	{∗110(1)108(4)107(9)106(17)105(20)104(18)103(12)102(19)101(10)100(8)99(6)98(3)97(1)∗}	97(1)
LFSR$_{15}$	Key_{17}	{∗109(2)108(4)107(12)106(18)105(17)104(22)103(18)102(12)101(11)100(5)98(4)97(2)95(1)∗}	95(1)
LFSR$_{15}$	Key_{18}	{∗109(1)108(4)107(15)106(13)105(25)104(22)103(11)102(15)101(6)100(9)99(3)98(2)97(1)96(1)∗}	96(1)
LFSR$_{15}$	Key_{19}	{∗109(1)108(2)107(12)106(14)105(15)104(23)103(14)102(17)101(9)100(5)99(5)98(2)97(1)96(4)95(1)92(2)91(1)∗}	91(1)
LFSR$_{15}$	Key_{20}	{∗109(2)108(4)107(8)106(15)105(24)104(29)103(13)102(9)101(9)100(6)99(5)97(2)96(2)∗}	96(2)
LFSR$_{15}$	Key_{21}	{∗109(3)108(3)107(10)106(24)105(10)104(18)103(20)102(15)101(7)100(7)99(3)98(5)97(1)96(1)94(1)∗}	94(1)
LFSR$_{15}$	Key_{22}	{∗108(6)107(9)106(17)105(23)104(22)103(17)102(5)101(12)100(8)99(2)98(2)97(4)95(1)∗}	95(1)
LFSR$_{15}$	Key_{23}	{∗109(2)108(2)107(8)106(12)105(27)104(20)103(12)102(15)101(7)100(4)99(9)98(5)97(4)96(1)∗}	96(1)
LFSR$_{15}$	Key_{24}	{∗110(1)108(6)107(14)106(17)105(17)104(20)103(18)102(9)101(8)100(5)99(5)98(1)97(4)95(2)94(1)∗}	94(1)
LFSR$_{15}$	Key_{25}	{∗109(2)108(4)107(19)106(13)105(21)104(18)103(14)102(10)101(10)100(6)99(4)98(5)95(2)∗}	95(2)
LFSR$_{15}$	Key_{26}	{∗109(1)108(8)107(6)106(6)105(30)104(14)103(14)102(13)101(17)100(6)99(8)98(2)97(1)96(1)95(1)∗}	95(1)
LFSR$_{15}$	Key_{27}	{∗109(1)108(10)107(10)106(17)105(16)104(8)103(21)102(13)101(14)100(9)99(4)98(3)97(1)95(1)∗}	95(1)
LFSR$_{15}$	Key_{28}	{∗109(1)108(5)107(13)106(22)105(20)104(17)103(14)102(15)101(4)100(10)99(3)98(3)96(1)∗}	96(1)
LFSR$_{15}$	Key_{29}	{∗110(1)108(9)107(10)106(13)105(22)104(21)103(13)102(12)101(10)100(6)99(7)98(1)97(3)∗}	97(3)