A Comprehensive Security Analysis of the TUAK Algorithm Set

Kalikinkar Mandal, Yin Tan, Teng Wu, and Guang Gong, *Fellow, IEEE*

Abstract

The authentication and key generation functions play a significant role to guarantee security and privacy in cellular mobile communications. MILENAGE is a set of authentication and key generation functions proposed by the 3^{rd} Generation Partnership Project (3GPP). Most recently, the 3GPP Task Force proposed a new set of authentication and key generation functions, called TUAK, as an alternative for the 4G LTE cellular mobile communications. The design of the TUAK functions makes use of the Keccak permutation, which is a core function in the SHA-3 hash function. One may expect that the TUAK functions may inherit attack resistance properties from that of the Keccak permutation. However, TUAK functions are obtained by assigning keys, different constants and padding bits at different input positions of the Keccak permutation. Therefore, a security analysis for the construction of the TUAK functions is needed. In this paper, for the first time, we present a comprehensive security analysis of the TUAK function set. Our main contribution is to provide a security analysis of TUAK's authentication and key derivation functions under various relevant cryptanalytic attacks. We consider powerful cryptanalytic attacks including differential attack, linear attack, distinguishing attack, boomerang attack and key-recovery attacks such as algebraic attacks and cube attacks. We conduct an experiment to find collisions on f_2 and f_5 functions and our experiment finds collisions on f_2 function and near-collision on f_5 function, due to the small output sizes. Our security analysis shows that the TUAK functions are resistant to the cryptanalytic attacks.

I. Introduction

In cellular mobile communication, the security and privacy are achieved through a series of algorithms that provide authentication (between the users and the authentication center) and generate various keys. Designing a framework for the authentication and key generating functions that was secure and flexible is one of the most important tasks. It is well known that EPS-AKA (see Appendix) is a protocol for the mutual authentication and key agreement. This protocol requires a series of functions, denoted by $f_1, f_2, \ldots, f_5, f_1^*, f_5^*$, to provide message authentication codes (MAC) and to generate keys. The 3^{rd} Generation Partnership Project (3GPP) proposed the MILENAGE function set to provide the choices of the aforementioned f_i in EPA-AKA. For the details of MILENAGE, the readers are referred to the TS 35.205 document of the 3GPP specification [1]. Most recently, for the higher security consideration in 3GPP systems, the 3GPP Task Force proposed a new algorithm set, called TUAK, containing seven security functions for authentication and key agreement/derivation, which is another choice of the f_i functions in EPS-AKS [2]. This offers the operators more methods to generate MACs and keys in cellular communication systems.

The TUAK algorithm set consists of eight security functions: $\text{TOP}_c, f_1, f_1^*, f_2, f_3, f_4, f_5, f_5^*$. The function TOP_c is used to encrypt TOP value preserved by the operator, f_1, f_1^* are used to generate message authentication codes, and the rests are designed to generate various keys

Y. Tan, T. Wu and G. Gong are with the Department of Electrical and Computer Engineering, University of Waterloo, Ontario, N2L 3G1 Canada. e-mail: {yin.tan, teng.wu, ggong}@uwaterloo.ca. K. Mandal is with Department of Electrical Engineering, University of Washington, Seattle, USA. e-mail: kmandal@uw.edu.

The work by Kalikinkar Mandal was done when he was a Post-doctor in the University of Waterloo.

used in cellular mobile communications. The design of TUAK makes use of the Keccak permutation of the SHA-3 hash function [7], [9]. In the last few years, the cryptographic properties of the Keccak permutations and cryptanalysis of the SHA-3 hash function have been extensively studied [3], [13], [17], [22], [31], [37], [48]–[50]. In the TUAK function set, different functions are constructed by assigning different inputs including padding and constants at different positions of the input to the Keccak permutation. This demands that TUAK functions should be carefully analyzed for avoiding inappropriate use of Keccak such that the security level does not decrease. To the best of our knowledge, there is no systematic cryptanalysis on the algorithms in TUAK. Our goal in this paper is to provide a comprehensive security analysis of the TUAK functions to assess the security levels of the authentication and key agreement functions of TUAK. In our analysis, we consider three powerful and different types of attacks, namely distinguishing attacks, collision attacks and key-recovery attacks. We analyze TUAK functions against differential attack, linear attack, distinguishing attack, boomerang attack, algebraic attack and cube attack and provide the complexities of these attacks. Moreover, we perform some experiments to analyze the cryptographic properties of the component functions of TUAK authentication function. According to our analysis, the complexity of the attacks shows that the TUAK functions are resistant to those three types of attacks.

The rest of the paper is organized as follows. In Section 2, we define some notations and review the design of the TUAK functions. In Sections 3 and 4, we present the security analysis of TUAK functions under distinguishing attacks, collision attacks and key-recovery attacks. In Section 5, some concluding remarks are provided.

II. Description of TUAK Function Set

The TUAK function set consists of eight functions, $\text{TOP}_c, f_1, f_1^*, f_2, f_3, f_4, f_5, f_5^*$, in the following we only introduce f_1. As mentioned in Introduction, the TUAK functions are constructed using the Keccak permutation. For the details of the Keccak permutation, the readers are referred to [7]. In this section, we describe the authentication function and the key derivation function of TUAK. Other functions are described in Appendix or can be found in [2]. We use the notations used in [2] to describe TUAK functions.

Notations

- The Keccak permutation Keccak-$f[1600]$ is denoted by Π, which takes a binary input of length 1600. The input bits are denoted by $\text{IN}[0], \ldots, \text{IN}[1599]$ and the output bits of Π are denoted by $\text{OUT}[0], \ldots, \text{OUT}[1599]$;
- ALGONAME is a fixed binary string of 56 bits. Its value can be found in [2];
- 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;

A. 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 subscriber key K (128 or 256 bits). The output value MAC can be 64, 128 and 256 bits. Precisely:

- The value of INSTANCE is:

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

- $\text{IN}[0] \ldots \text{IN}[255] = \text{TOP}_c[255] \ldots \text{TOP}_c[0]$;
- $\text{IN}[256] \ldots \text{IN}[263] = \text{INSTANCE}[7] \ldots \text{INSTANCE}[0]$;
- $\text{IN}[264] \ldots \text{IN}[319] = \text{ALGONAME}[55] \ldots \text{ALGONAME}[0]$;
- $\text{IN}[320] \ldots \text{IN}[447] = \text{RAND}[127] \ldots \text{RAND}[0]$;
- $\text{IN}[448] \ldots \text{IN}[463] = \text{AMF}[15] \ldots \text{AMF}[0]$;
- $\text{IN}[464] \ldots \text{IN}[511] = \text{SQN}[47] \ldots \text{SQN}[0]$;
- $\text{IN}[512] \ldots \text{IN}[767] = K[255] \ldots K[0]$ 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 $640 \leq i \leq 767$ 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 $1088 \leq i \leq 1599$.

A high-level overview of the input assignment is provided in Figure 7. 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

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

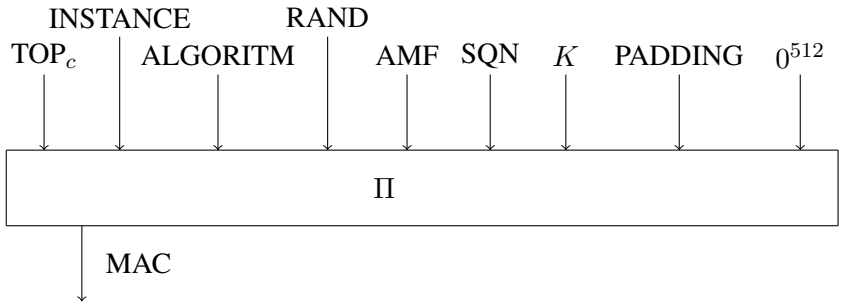

Fig. 1: The f_1 function of TUAK for generating MAC

III. Security Analysis of TUAK Algorithms

In this section, we provide the security analysis for the algorithms in TUAK under the differential attack, linear attack, birthday attack, distinguishing attack and boomerang attack. For each of the aforementioned attack, we will provide the complexity of the attack.

A. Differential cryptanalysis

As mentioned in Section II, the design of the algorithms in TUAK is based on the Keccak permutation. This design enables us to relate the linear and differential attack on the algorithms in TUAK to the corresponding attack on Keccak. Indeed, it is not difficult to see that a differential attack on the algorithms in TUAK will be directly the attack on Keccak. We first give the below table which summarizes the recent differential attacks on the reduced-round Keccak.

TABLE I: Known differential 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)	-	-	[27], [37]
4 (practical)	4 (practical)	-	-	[22]
-	5 (2^{115})	3 (practical), and 4 (2^{147})	3 (practical)	[19]

From Table I, we see that the complexity of the differential attack on 5-round Keccak is already impractical. In the following, we provide a lower bound of the complexity for the full-round differential attack on TUAK's algorithms. Our analysis uses the results in [17] 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 II presents the lower bound of a differential trail of round-reduced Keccak and the same lower bounds also hold for an algorithm in TUAK.

TABLE II: Lower bound on the complexity of differential attack on functions in TUAK

Round	Lower Bound on $wr(Q)$	Lower bound of Complexity	Reference
3	32	2^{32}	[17]
6	74	2^{74}	[17]
24	296	2^{296}	[17]

B. Linear cryptanalysis

Similar to the differential attack, a linear attack on an algorithm in TUAK will lead to the linear attack on Keccak. In [7], the weight of the linear trails of up to 6-round Keccak (see [7, Table 3.3]). It is well known that 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 [7, page 24]) of the linear trail Q. According to Table 3.3 of [7], the minimum weight of the linear trail is 46 for 4-round Keccak. The estimation on the lower bound of the weight of a linear trail on 24-round Keccak is then 276. Since a linear trail in Keccak could be a linear trail in TUAK's algorithms, and a linear trial of TUAK's algorithms must be the one of Keccak, the lower bound of the complexity of the linear attack on TUAK's algorithms is 2^{276}.

C. Zero-sum distinguisher of TUAK

Zero-sum distinguisher against Keccak was proposed in [49], and this attack was improved in [31]. These two zero-sum distinguishers of Keccak are based on the following result.

Theorem 1 ([31], [49])**.** *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 $\mathbb{F}_{2^{n_0}}$. 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 and R^i is the i-th composition of R.

Corollary 1. *Let $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$ be the round function of* Keccak *permutation, where $\iota, \chi, \pi, \rho, \theta$ are the component functions defined in [7]. Then for any function G from $\mathbb{F}_{2^{1600}}$ to itself, we have*

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

Now, for each algorithm f in Keccak, since it has constant values in certain bits, we may regard it as a permutation by restricting the Keccak permutation on some subspace of $\mathbb{F}_2^{1600}$. Writing $f_i = \texttt{Keccak-}f|_{\mathbb{F}_2^m} : \mathbb{F}_2^m \to \mathbb{F}_2^m$, 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.

Applying Corollary 1, the complexities of TUAK's zero-sum distinguisher are provided in Table V. In the table, the numbers in bold means the complexity are better than the exhaustive search. From Table V we can observe that: 1) 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.

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

No of round	UB (forward)	UB (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

TABLE IV: 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	UB (forward)	UB (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

TABLE V: 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 V 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.

D. Birthday attack

A birthday attack is a generic attack that can be applied to any cryptosystem and which has been discussed in [36]. 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 TOP_C are constant. Recall that the TOP_C function accepts the keys of length 128 bits and 256 bits and outputs TOP_c of length 256 bits. 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 RAND 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 VI.

1) Practical (near-)collision on f_2-32: Here we present near-collisions and collisions on f_2 with output 32 bits. We implemented a birthday-type attack to find (near-)collisions of f_2.

TABLE VI: 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	$O(2^{128})$	$O(2^{64})$	$O(2^{128})$	$O(2^{64})$	$O(2^{128})$	$O(2^{24})$	$O(2^{24})$

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 or collision with Hamming weight one and zero, respectively, for f_2. For $P \leq 2^{14}$, we have found a number of near-collisions and 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. We believe the reason behind successfully finding a collision or near-collision is due to the short length of RES.

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

Fig. 2: Near-collision of f_2-32

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

Fig. 3: Collision of f_2-32

2) Near-Collision on f_5: Although the random number RAND in the protocol in Figure 5 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.

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

E. Differential distinguishing attack

Kim et al. [30] proposed a differential distinguishing attack on a HMAC-based message authentication code, which exploits the differential property of the underlying hash function to forge a MAC. The construction of TUAK is different from the construction of HMAC attacked in [30]. We analyse 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 [8]. According to the CICO problem, it is hard to obtain a good differential characteristic with probability greater than $2^{-\frac{|M|}{2}}$ where M is the length of the output of a TUAK function. Moreover, in Section III-A, 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 $2^{\frac{M}{2}}$.

F. Boomerang attack

The boomerang attack was proposed by Wagner against block ciphers by finding and combining short differential paths, instead of using a long differential path [40]. 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. Below we argue why the TUAK algorithm set is resistant to the boomerang attack.

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 $\mathtt{DP}_s : \Delta \rightarrow \Delta'$ with probability p_s is for Π_1 and another differential path $\mathtt{DP}_e : \nabla' \rightarrow \nabla$ 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. $P_1 = \text{TOP}_c||\text{INSTANCE}||\text{ALGONAME}||\text{RAND}||\text{AMF}||\text{SQN}||\text{K}||\text{PADDING}||0^{512}$.

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

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 $\mathtt{DP}_s$ $\mathtt{DP}_e$ 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 [9]. As a result, it is hard to find good differential characteristics for Π_2 as well as its inverse. According to [17], 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. Moreover, the TUAK algorithm set will resist the boomerang attack of Joux and Peyrin in [29] because of hardness of finding a good auxiliary differential characteristic with probability greater than 2^{-296}.

IV. Security Analysis on Key Recovery Attack on TUAK Algorithms

In this section, we analyze the security of TUAK against the key recovery attacks by considering key-search reduction attack, algebraic attack, cube attack, interpolation attack and key-recovery attack on MAC. We provide the complexity of the attacks to recover the secret key. We perform an experiment to measure the cryptographic properties of the component functions constructed in multi-output filtering model as introduced in [54].

A. Key-search complexity reduction with structural property of TUAK

From the definitions of $f_2 \sim f_5$, we can observe that the inputs of $f_2 \sim f_5$ are 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, and IK for a random number RAND and she does not know 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. However, when RES and IK are known to the adversary, she can easily decide the correct key by computing RES and IK for a key and RAND and those 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. Our observation suggests to have different INSTANCE values for the functions $f_2 \sim f_5$. 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 $O(2^{|K|})$.

B. Interpolation attack

Interpolation attack on block ciphers was proposed by Jakobsen and Knudsen in [28]. The complexity of an interpolation attack depends on the degree of the polynomial approximation and the number of nonzero terms in the polynomial approximation [28], [43]. 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 III and IV, 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.

C. Experiment on component functions

In this section, we provide an 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 of the Keccak permutation. 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.

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. For this choice of input assignment to TUAK, we can construct a set of component functions using Eq. (1) and we denote the set of component functions by $\{g_0, ..., g_{255}\}$

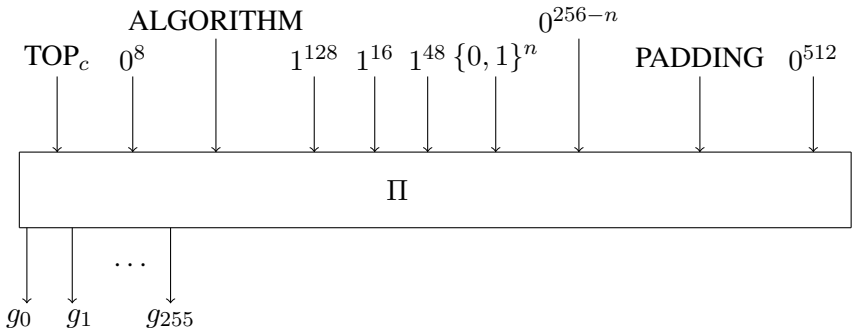

Fig. 4: Component functions of TUAK in n variables

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; 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. The results for the algebraic degree and the number of terms of component functions can be found in Tables VII. The nonlinearity of a component function in 8 variables lies between 94 and 108. For the details on nonlinearities of the component functions of TUAK for $n = 9, ..., 13$, see our technical report [61].

TABLE VII: 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	$3957 \sim 4213$

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 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**.

D. Algebraic attack

Algebraic attack [15] 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 linearisation, 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 user 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 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: 1) using only one function f_i for different random numbers; 2) using a set of functions f_i's for different random numbers; 3) 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 IV-C, our results show that the algebraic degree and algebraic immunity of a component function of TUAK is maximum for $n = 8, ...,$ 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 linearisation 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. The time complexity of recovering the key for a TUAK algorithm is $O(2^{|K| \cdot w}), |K| = 128$ and 256 when the length of the key is 128 and 256, respectively.

E. Cube attack

Cube attack [18] is a key-recovery attack that can be applied to any cryptosystem, and a cube tester is used to detect the nonrandom behaviour of a cryptographic function [4]. 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 update of a cryptoalgorithm grows slowly [18]. 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 [32], 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 [23] 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 IV-C 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.

F. Slide attacks

Slide attacks, proposed by Biryukov and Wagner [11], 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 CK, integrity key IK, or anonymity key AK 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 [26] proposed a slide attack on sponge-function like hash structures and the attack can also be applied to message authentication codes 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 the above MAC, the 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 [26]. 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.

G. 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 [14] presented a partial key recovery attack on HMAC/NMAC-MD4/MD5/SHA0 using collisions of MD4, MD5, and SHA0. Later, Fouque et al. [24] proposed a key recovery attack on HMAC/NMAC-MD4 that can recover the full key of MAC. Both attacks rely on good differential paths for the underlying compression function that lead to collisions.

We show the attack resistance of properties of the TUAK algorithm set against the above two attacks. Note that the basic requisite for the above two attacks is a good differential characteristic of the underlying hash function of HMAC. To apply 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 the above differential-style key-recovery attacks. 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 [17], 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}. Thus, the above differential-style key recovery attack cannot be mounted efficiently to the TUAK algorithm set.

H. 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 [33], [41], [42]. An extension attack may also help to mount a distinguishing attack and a key recovery attack [14], [24]. 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 [8].

V. CONCLUSIONS

In this paper, we analyzed the authentication function and key derivation/agreement functions $f_1 \sim f_5$, f_1^* and f_5^* of TUAK by considering several differential-style attacks, birthday attacks, interpolation attacks, algebraic attack, cube attack, slide attacks and extension attacks. Our analysis shows that, due to good differential attack resistance properties of the Keccak permutation, the TUAK function set is resistant to collision attacks and differential-style key-recovery attacks. Applying a birthday-style attack, we are able to find collisions on f_2-32 function and near-collision on f_5-48 function. We believe that the successful collision finding depends on the short output size of the algorithm and this can easily be avoided by increasing the output size. We also analyzed the cryptographic properties of the component functions of the TUAK functions. We emphasize that an attacker may recover the key of the TUAK algorithm set with complexity less than that of the exhaustive search by exploiting the structural properties of $f_2 \sim f_5$ as they use the same input to derive the cipher key, integrity key and anonymity key. Finally, our analysis demonstrates that the time complexity of other key recovery attacks is not better than the exhaustive key search.

ACKNOWLEDGEMENT

The research is partially supported by GSMA and partially supported by NSERC CRD.

REFERENCES

[1] 3GPP 3GPP TS 35.206 V6.0.0 Technical Report; 3rd Generation Partnership Project; 3G Security; Specification of the MILENAGE algorithm set: An example algorithm Set for the 3GPP Authentication andKey Generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 2: Algorithm specification (Release 6). http://www.etsi.org/deliver/etsi_ts/135200_135299/135206/06.00.00_60/ts_135206v060000p.pdf

[2] 3GPP TS 35.231 V12.1.0 Technical Report; 3rd Generation Partnership Project; LTE Security; Specification of the TUAK algorithm set: A second example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 1: Algorithm specification (Release 12) http://www.etsi.org/deliver/etsi_ts/135200_135299/135231/12.01.00_60/ts_135231v120100p.pdf

[3] J.-P. Aumasson and W. Meier. 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.

[4] 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.

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

[6] M. Bellare, A. Desai, E. Jokipii, P. Rogaway. 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)

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

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

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

[10] E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems, Advances in Cryptology, pp. 2 – 21, 1991.

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

[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] 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.

[15] 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.

[16] 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.

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

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

[19] 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.

[20] I. Dinur, O. Dunkelman, and A. Shamir. 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/

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

[22] 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.

[23] 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/

[24] 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.

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

[26] 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.

[27] 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.

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

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

[30] 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 (extended abstract). In SCN 2006 (2006), Prisco R. D., Yung M., (Eds.), LNCS 4116, pp. 242 – 256, Springer, 2006.

[31] 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

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

[33] 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.

[34] 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.

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

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

[37] 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.

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

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

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

[41] 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.

[42] 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.

[43] 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.

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

[45] 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.

[46] D.J. Bernstein. Second preimages for 6 (7? (8??)) rounds of Keccak?. http://ehash.iaik.tugraz.at/uploads/6/65/NIST-mailing-list_Bernstein-Daemen.txt (2010)

[47] C. Boura, and A. Canteaut. On the influence of the algebraic degree of F^{-1} on the algebraic degree of $G \circ F$. IEEE Trans. on Information Theory, 59(1), 691-702 (2013)

[48] C. Boura, A. Canteaut, C. De Cannière. Higher-order differential properties of Keccak and Luffa. In: Joux, A. (ed.) FSE 2011, LNCS, vol. 6733, pp. 252-269, Springer, Heidelberg (2011)

[49] C. Boura, and A. Canteaut. Zero-sum distinguishers for iterated permutations and application to Keccak-f and Hamsi-256. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2011, LNCS, vol. 6544, pp. 1-17, Springer, Heidelberg (2011)

[50] A. Duc, J. Guo, T. Peyrin, and L. Wei. Unaligned rebound attack: application to Keccak. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 402-421, Springer, Heidelberg (2012)

[51] D. Khovratovich, A. Biryukov, and I. Nikolic. Speeding up collision search for byte-oriented hash functions. In: Fischlin, M. (ed.) CT-RSA 2009, LNCS, vol. 5473, pp. 164-181, Springer, Heidelberg (2009)

[52] D. Khovratovich. Cryptanalysis of hash functions with structures. In: Jacobson, M.J., Jr., Rijmen, V., and Safavi-Naini, R. (eds.) SAC 2009, LNCS, vol. 5867, pp. 108-125, Springer, Heidelberg (2009)

[53] L.R. Knudsen. Truncated and higher order differentials. In: Preneel, B. (ed.) FSE 1995, LNCS, vol. 1008, pp. 196-211, Springer, Heidelberg (1995)

[54] G. Gong, K. Mandal, Y. Tan, and T. Wu. On the Multi-output Filtering Model and Its Applications. Cryptology ePrint Archive, Report 2014/542 (2014), https://eprint.iacr.org/

[55] D.R. Stinson. Cryptography - theory and practice. CRC Press (1995)

[56] X. Wang, H. Yu, Y.-L. Yin. Efficient collision search attacks on SHA-0. In: Shoup, V. (ed.) CRYPTO 2005, LNCS, vol. 3621, pp. 1-16, Springer, Heidelberg (2005)

[57] X. Wang, Y.L. Yin, H. Yu. Finding collisions in the full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005, LNCS, vol. 3621, pp. 17-36, Springer, Heidelberg (2005)

[58] H. Yu, and X. Wang. Near-collision attack on the compression function of dynamic SHA2. Cryptology ePrint Archive, Report 2009/179 (2009), http://eprint.iacr.org/2009/179

[59] NIST, the SHA-3 competition (2007-2012). http://csrc.nist.gov/groups/ST/hash/sha-3/index.html

[60] Y. Tan. Computational results on Keccak. In www.ytan.me.

[61] G. Gong, K. Mandal, Y. Tan, and T. Wu, Security Assessment of TUAK Algorithm Set, Technical Report to 3GPP, can be download from http://www.3gpp.org/ftp/tsg_sa/WG3_Security/TSGS3_76_Sophia/TdocList_2014-08-29_16h55.htm with the document No. S3-142035 (116 pages).

APPENDIX

A. The EPS-AKA Protocol

EPS-AKA is a mutual authentication and key agreement protocol of the 4G LTE network. The *EPS-AKA* protocol uses a series of functions, named f_i, $1 \leq i \leq 5$ defined below. The purpose of the 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. The authentication and key-agreement process is shown in Figure 5.

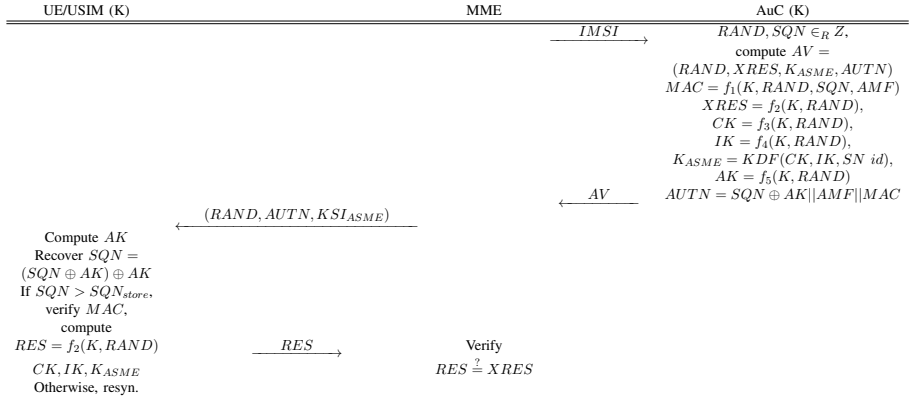

Fig. 5: The EPS-AKA protocol

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

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

where $RAND_i$ 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 5, 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.

VI. 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

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

- $\text{IN}[0]\ldots\text{IN}[255] = \text{TOP}[255]\ldots\text{TOP}[0]$;
- $\text{IN}[256]\ldots\text{IN}[263] = \text{INSTANCE}[7]\ldots\text{INSTANCE}[0]$;
- $\text{IN}[264]\ldots\text{IN}[319] = \text{ALGONAME}[55]\ldots\text{ALGONAME}[0]$;
- $\text{IN}[i] = 0,$ for $320 \leq i \leq 511$;
- $\text{IN}[512]\ldots\text{IN}[767] = K[255]\ldots K[0]$ 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 $1088 \leq i \leq 1599$.

Figure 6 depicts an overview of an input assignment to the TOP_C function. The TOP_C function is given by

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

with

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

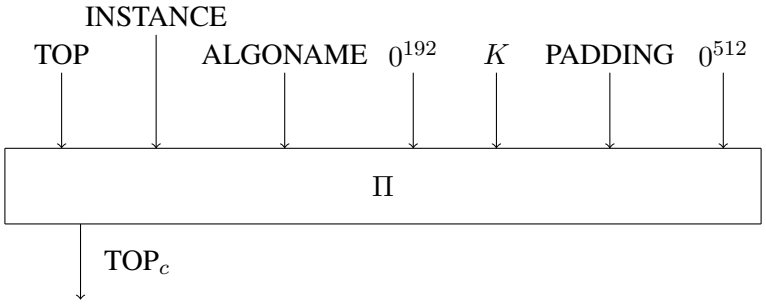

Fig. 6: The TOP_c function of TUAK

A. 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:

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

- $\text{IN}[0] \dots \text{IN}[255] = \text{TOP}_c[255] \dots \text{TOP}_c[0]$;
- $\text{IN}[256] \dots \text{IN}[263] = \text{INSTANCE}[7] \dots \text{INSTANCE}[0]$;
- $\text{IN}[264] \dots \text{IN}[319] = \text{ALGONAME}[55] \dots \text{ALGONAME}[0]$;
- $\text{IN}[320] \dots \text{IN}[447] = \text{RAND}[127] \dots \text{RAND}[0]$;
- $\text{IN}[i] = 0, 448 \leq i \leq 511$;
- $\text{IN}[512] \dots \text{IN}[767] = K[255] \dots K[0]$ if the length of K is 256 bits ;
- $\text{IN}[512] \dots \text{IN}[639] = K[127] \dots K[0]$ 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:

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

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

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

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

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

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

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

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

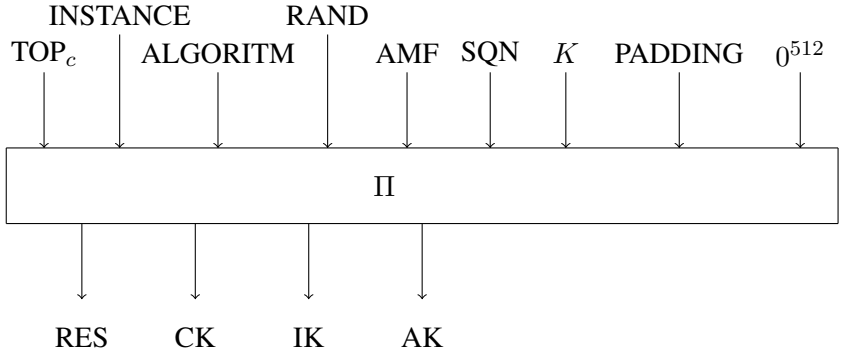

Fig. 7: The f_i function of TUAK for generating RES, CK, IK, AK

B. Description of f_5^*

or 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

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

C. Summary of TUAK algorithms

We denote by f_i-M the f_i function/algorithm with output M bits. We denote the TUAK algorithm set by

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

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,

$$f_i : \{0,1\}^{1600} \rightarrow \{0,1\}^M \text{ where } M = 32, 64, 128, \text{ or } 256.$$

$$(g_0(x), g_1(x), ..., g_{M-1}(x)) = f(x), x \in \mathbb{F}_2^{1600}, f \in \mathcal{TUAK}, \quad (1)$$

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.