Key-recovery attacks against the MAC algorithm Chaskey*

Chrysanthi Mavromati

[1] Capgemini-Sogeti, R&D Lab, France
[2] Université de Versailles Saint-Quentin-en-Yvelines, Laboratoire PRISM, France
chrysanthi.mavromati@sogeti.com

Abstract. Chaskey is a Message Authentication Code (MAC) for 32-bit microcontrollers proposed by Mouha *et. al* at SAC 2014. Its underlying blockcipher uses an Even-Mansour construction with a permutation based on the ARX methodology. In this paper, we present key-recovery attacks against Chaskey in the single and multi-user setting. These attacks are based on recent work by Fouque, Joux and Mavromati presented at Asiacrypt 2014 on Even-Mansour based constructions. We first show a simple attack on the classical single-user setting which confirms the security properties of Chaskey. Then, we describe an attack in the multi-user setting and we recover all keys of 2^{43} users by doing 2^{43} queries per user. Finally, we show a variant of this attack where we are able to recover keys of two users in a smaller group of 2^{32} users.

Keywords: Message Authentication Code, Collision-based cryptanalysis, ARX, Even-Mansour, Chaskey, Multi-user setting.

1 Introduction

A Message Authentication Code (MAC) algorithm is a basic component in many cryptographic systems and its goal is to provide integrity and data authentication. For this, it takes as input a message M and a n-bit secret key K, and outputs a tag τ which is usually appended to M. In general, MAC algorithms are built from universal hash functions, hash functions or block ciphers. The first security property of a MAC is that it should be impossible to recover the key K faster than 2^n operations (key recovery attack). It should also be impossible for an attacker to forge a tag for any unseen message without knowing the key (MAC forgery). Here, the attacker selects a message and simply guesses the correct MAC value without any knowledge of the key. The probability that the guess will be correct is 2^{-t} where t is the size of the tag τ. However, such attacks can be avoided by making the tag sufficiently large. MACs based on universal hash functions compute the universal hash of the message to authenticate and then

* Final version submitted by the author and published in the proceedings of SAC 2015.

encrypt this value. MAC algorithms based on block ciphers are usually given as a mode of operation for the block cipher. Typically, CBC-MAC is very similar to the CBC encryption mode of operation and is one of the most commonly used MACs.

The implementation of a MAC algorithm on a typical microcontroller is a challenging issue as many problems may appear. More precisely, the use of a MAC based on a hash function or a block cipher might have a negative impact on the speed of the microcontroller due to the computational cost of the operations required by the underlying functions.

Recently, at SAC 2014, a new MAC algorithm named Chaskey has been introduced by Mouha *et al.* [8]. It is a permutation-based MAC algorithm and its underlying permutation relies on the ARX design. The designers claim that Chaskey is a lightweight algorithm which overcomes the implementation issues of a MAC on a microcontroller. The construction of Chaskey is based on some simple variants of the CBC-MAC proposed by Black *et al.* [3]. As CBC-MAC is not secure when used with messages of variable length, these variants were designed to authenticate arbitrary length messages. Alternatively, Chaskey can also be seen as an iterated Even-Mansour construction where the same subkey is xored in the input and output of the last permutation round.

In Asiacrypt 2014, Fouque *et al.* [6] presented new techniques to attack the Even-Mansour scheme in the multi-user setting. Based on this work, we present here some key-recovery attacks against the MAC algorithm Chaskey in the single and multi-user setting.

The paper is organized as follows: in Section 2 we recall the necessary background, present the multi-user setting, the Even-Mansour scheme and the general principle of collision-based attacks using the distinguished points method, in Section 3 we present the main specifications of the Chaskey MAC algorithm and finally, in Section 4, we describe new attacks against Chaskey.

2 Preliminaries

2.1 The multi-user setting

The security of most schemes in cryptography is usually studied when we have a single recipient of encrypted data: the single-user model. However, this setting ignores an important dimension of the real world where there are many users, who are all using the same algorithms, but each one has its own key. In this setting, the attacker tries to recover all or fraction of keys more efficiently than the complexity of the attack in the single-user model times the number of users. This can be done by amortizing the cost of the attack among the users.

In [7], Menezes studies the security of MAC algorithms in the multi-user setting. He shows that the security degrades when we pass from one to many users. The scenario attack against MAC algorithms in the multi-user setting can be seen as follows. Let $H_K : \{0,1\}^* \to \{0,1\}^t$ be a family of MACs where $K \in \{0,1\}^k$. In the single-user model, it should be hard for an attacker who has access to an oracle H_K to generate a valid message-tag pair (m, τ) without knowing the key K. In the multi-user setting, we suppose that we have L users with keys $K^{(i)} \in \{0,1\}^k$ where $0 \leq i \leq (L-1)$. Then for an attacker who has access to oracles for $H_{K^{(i)}}$, it should be difficult to produce a triplet (i, m, τ).

2.2 Collision-based attacks using the distinguished point technique against the Even-Mansour scheme

The distinguished point technique. The distinguished point technique allows to find collisions in a very efficient way. For this, we first have to define a function f on a set S of size N and then define a distinguished subset S_0 of S with $\mathcal{D}$ distinguished points. The distinguished points should be easy to recognize and generate. For example, we can choose our distinguished points to be n-bits values with d zeros at the end. So, in a set of cardinality $N = 2^n$ our distinguished subset contains $\mathcal{D} = 2^{n-d}$ elements. Starting from a random point $x_0 \in S$ we build chains by evaluating the function f:

$$x_{i+1} = f(x_i).$$

When a distinguished point is detected, *i.e.* $x_\ell \in S_0$, the construction of the chain stops. To be able to recover the chain we need to store the starting point x_0, the distinguished point x_ℓ and the length of the chain ℓ which corresponds to the number of iterations of the function f. Two chains that pass through the same point necessarily end at the same distinguished point. To detect a collision we use the inverse result: two chains that end at the same distinguished point necessarily merge at some point unless one chain is a subchain of the other. Once a collision in the distinguished points set is detected, the real collision can easily be recovered. We assume that we have two colliding chains of length ℓ and $\ell^{'}$ and that $\ell \geq \ell^{'}$. Then, starting from the longer chain, we rebuild the chain by taking exactly $\ell - \ell^{'}$ steps. From that point, it now suffices to build both chains in parallel until a collision is reached.

Collision-based attacks against Even-Mansour. The Even-Mansour scheme is a minimalistic very efficient design of a block cipher proposed at Asiacrypt 91 [5]. The main idea is the construction of a keyed permutation family Π_{K_1,K_2} by a public permutation π that operates on n-bit values $(N = 2^n)$:

$$\Pi_{K_1,K_2} = \pi(m \oplus K_1) \oplus K_2,$$

where m is the plaintext and K_1 and K_2 are the two whitening keys. There is also a simpler version presented by Dunkelman *et al.* [4], the *Single-key Even-Mansour*, where $K_1 = K_2$. Even and Mansour showed that this simple block-cipher is secure up to $\mathcal{O}(2^{n/2})$ queries of the adversary to the keyed permutation Π and to the public permutation π. It has also been proved [4] that the Single-key Even-Mansour has the same security bound as the original version.

In [6], Fouque *et al.* describe a new technique for collision-based attacks using the distinguished point technique. They use this method to form attacks against the Even-Mansour scheme in the multi-user setting. The main idea is to find a collision between two chains, one constructed from the public permutation π and one from the keyed permutation Π. For this, as we cannot use permutations to detect collisions, we have to construct two functions F and f, one based on Π and one based on π. In previous attacks on Even-Mansour [2, 4], these functions have been constructed by using the Davies-Meyer construction: $F(m) = \Pi(m) \oplus \Pi(m \oplus \delta)$ and $f(m) = \pi(m) \oplus \pi(m \oplus \delta)$. However, these functions cannot be used with the distinguished point technique as chains built by F and f defined as previously can eventually cross but they cannot merge as they consist of evaluations of two different functions. As a consequence, they use a different definition of F and f to solve this problem:

$$F(m) = m \oplus \Pi(m) \oplus \Pi(m \oplus \delta) \quad \text{and} \quad f(m) = m \oplus \pi(m) \oplus \pi(m \oplus \delta).$$

For two messages m and $m^{'}$ such as $m^{'} = m \oplus K_1$, they remark that $F(m^{'}) = f(m) \oplus K_1$ and so they get two *parallel* chains, *i.e.* two chains that have a constant difference between them.

They also remark that, for two different users i and j, two chains constructed by using $F_{\Pi}^{(i)}$ and $F_{\Pi}^{(j)}$, where $F_{\Pi}^{(i)}(m) = m \oplus \Pi^{(i)}(m) \oplus \Pi^{(i)}(m \oplus \delta)$ and $F_{\Pi}^{(j)}(m) = m \oplus \Pi^{(j)}(m) \oplus \Pi^{(j)}(m \oplus \delta)$, can also become parallel and their constant difference would be equal to the XOR of the users keys, *i.e.* $K_1^{(i)} \oplus K_1^{(j)}$.

To attack Even-Mansour using the distinguished point technique in the multi-user setting, they build a set of chains for the public user using the function f and a small number of chains for every user by using the keyed permutation F. Whenever a collision $F^{(i)}(x) = F^{(j)}(y)$ for two points x and y, where $x = y \oplus K_1$, is detected between two users U_i and U_j, it yields $K_1^{(i)} \oplus K_1^{(j)}$. From these collisions it is possible to construct a graph whose vertices are the users and the edges represent the xor of the first keys of the users with two colliding chains, *i.e.* $K_1^{(i)} \oplus K_1^{(j)}$. When enough edges are present a giant component appears in the graph. Then, it suffices to find a single collision $F^{(i)}(x) = f(y)$ between a user and the public user to reveal all keys of the users in the giant component.

One of the main ideas of their technique is to detect parallel chains by simply testing if $\pi(y) \oplus \pi(y \oplus \delta) = \Pi(x) \oplus \Pi(x \oplus \delta)$ for two distinguished point x and y where $x = y \oplus K_1$. This does not add any extra cost, as values of $\pi(y) \oplus \pi(y \oplus \delta)$ and $\Pi(x) \oplus \Pi(x \oplus \delta)$ are needed to calculate the next element of the chain. Also, it does not require to go back and recompute the chains to detect the merging points.

3 The MAC algorithm Chaskey

Chaskey was proposed at SAC 2014 by Mouha *et al.*. It is a permutation-based MAC algorithm and its underlying permutation is based on the ARX design. Its design is similar to the permutation of the MAC algorithm SipHash [1]. However, in Chaskey, a state of 128-bits (instead of 256-bits in SipHash) is used which is decomposed in 4 words of 32-bits (instead of 64-bits in SipHash). Also, different rotation constants are used.

Chaskey takes as input a message M of arbitrary size and a 128-bit key K. The message M is split into ℓ blocks $m_1, m_2, \ldots, m_\ell$ of 128 bits each. If the last block is incomplete, a padding is applied. It outputs the t-bit tag τ (where $t \leq n$) that authenticates the message M. The underlying function is a permutation constructed using the ARX design.

Two subkeys K_1 and K_2 are generated from K as follows: K_1 is equal to the result of the multiplication by 2 (binary notation) of K and K_2 is equal to the multiplication by 2 of K_1. In general, we define $K_1 = \alpha K$ and $K_2 = \alpha^2 K$. To define multiplication in $GF(2^n)$, we need to specify the irreducible polynomial $f(x)$ of degree n that defines the representation of the field. The designers of Chaskey choose their irreductible polynomial to be $f(x) = x^{128} + x^7 + x^2 + x + 1$.

Chaskey operates in $\ell + 3$ steps. On the first step, we xor the first block m_1 with the key K. On the next step, we process the previous result through the permutation π and we xor the output with the next block. This procedure is repeated $\ell - 1$ times until all blocks are being processed. If the last block m_ℓ, which has been xored at the end of the $\ell - 1$ step, is a complete block, then, on the ℓ-th step, we simply xor the key K_1. If it is an incomplete block, it should be padded with $10^{n-|m_\ell|-1}$ before being used. Then, on the ℓ-th step, the key K_2 will be used instead of K_1. Then, the state will pass through the permutation π and will be xored with K_1 if m_ℓ is complete or with K_2 if incomplete. Finally, the t least significant bits are selected to be used as the tag τ. The whole procedure can be seen on Figure 1.

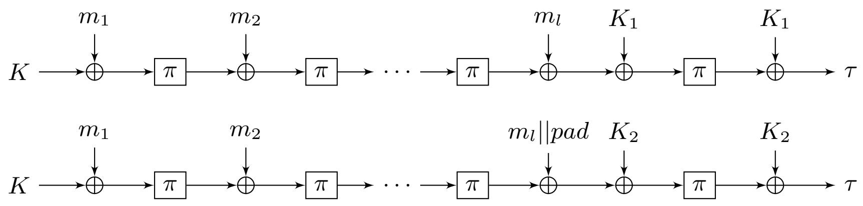

Fig. 1. The Chaskey MAC algorithm. First line when $|m_\ell| = n$ and second line when $0 \leq |m_\ell| \leq n$ where $pad = 10^{n-|m_\ell|-1}$.

There is also a variant of Chaskey, called Chaskey-B, where Chaskey can be seen as a MAC algorithm based on the Even-Mansour [5] block cipher. The authors define the block cipher E as $E_{X||Y}(m) = \pi(m \oplus X) \oplus Y$ and so it suffices to evaluate recursively the function $h_{i+1} = E_{K||K}(h_i \oplus m_i)$ for $i = 1, ..., l-1$ and $h_1 = 0^n$. If the last block is complete, we calculate $h_\ell = E_{K \oplus K_1 || K_1}(h_\ell \oplus m_\ell)$. If not, we calculate $h_\ell = E_{K \oplus K_2 || K_2}(h_\ell \oplus m_\ell || 10^{n-|m_\ell|-1})$. Finally, the tag τ is equal to the t least significant bits. However, in this case, we can easily see that the key K, which is XORed after the application of the permutation π, vanishes at each iteration of the block cipher E. As a result, the key K intervenes only on the first block and the keys K_1 or K_2 on the last block:

$$\tau = \pi(\pi(\pi(\pi(m_1 \oplus K) \oplus m_2) \oplus \ldots) \oplus m_l \oplus K_1) \oplus K_1.$$

The permutation π consists of 8 rounds of a function that follows the ARX design: addition modulo 2^{32}, bit rotations and XOR. For constructing this round function, the authors use the same structure as SipHash [1] but instead of 64-bit words they are using words of 32 bits and different rotation constants. They consider that 8 applications of the round function is secure but they suggest that the 16 rounds version (Chaskey-LTS: long term security) should also be implemented in case of security issues.

The authors prove that Chaskey is secure up to $D = 2^{n/2}$ chosen plaintexts and $T = 2^n/D$ queries to π or π^{-1}.

4 Collision-based attacks against Chaskey

In this section, we show that the collision-based attack described in [6] can be applied on Chaskey. Furthermore, we show that variants of this attack can be applied in the case of Chaskey. All attacks can be performed when we use single-block messages. Chaskey then becomes an Even-Mansour cipher:

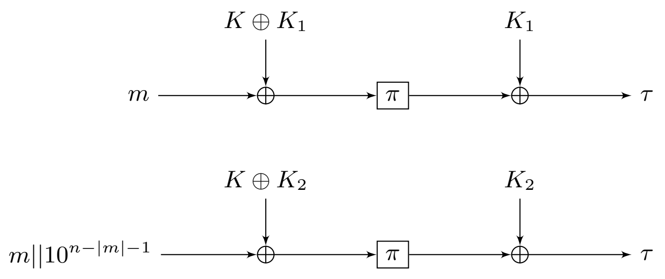

Fig. 2. Single-block messages in Chaskey

The main idea of all attacks is to build chains by using a function based on Chaskey and then search for collisions between them. For this, a function that can be used to build chains should be defined. However, in the case of Chaskey, for every user we can build two different chains depending on the subkey that we are using.

To build the chains needed, we define the functions:

$$f_s(M) = K_s \oplus \pi(M \oplus (K_s \oplus K))$$

$$F_{f_s}(M) = f_s(M) \oplus f_s(M \oplus \delta) \oplus M$$

where K is the users key, K_s with $s \in \{1,2\}$ represents the two subkeys generated as mentioned in section 3 and δ is an arbitrary but fixed non zero constant.

In the multi-user setting we assume that L different users are all using the Chaskey MAC algorithm based on the same public permutation π. Each user U_i, with $0 \leq i \leq L$, chooses its own key $K^{(i)}$ at random and independently from all the other users and generates K_s with $s \in \{1,2\}$. We define the functions $f_s(M)$ and $F_{f_s}(M)$ as above and we also define the function $F_\pi(M)$ for the public user as follows:

$$F_\pi(M) = M \oplus \pi(M) \oplus \pi(M \oplus \delta).$$

We remark that for two plaintexts M and $M^{'}$ where $M^{'} = M \oplus K_s \oplus K$, we have $F_{f_s}(M \oplus K_s \oplus K) = F_\pi(M) \oplus (K_s \oplus K)$. So, two chains based on functions F_{f_s} and F_π may become parallel.

4.1 Key-recovery attack in the single-user setting

In this section, we show that an attack with complexity 2^{64} can be applied in the classical single-user scenario. This attack does not contradict the security bound showed in the original paper of Chaskey and has similar complexity with possible slide attacks based on [2]. However, with this attack we show a simple application on Chaskey of the parallel chains detection technique.

The attack is described below:

1. Create two chains constructed by using both:

   $F_{f_1}(M) = f_1(M) \oplus f_1(M \oplus \delta) \oplus M$ and $F_{f_2}(M^{'}) = f_2(M^{'}) \oplus f_2(M^{'} \oplus \delta) \oplus M^{'}$

   until $f_1(M) \oplus f_1(M \oplus \delta)$ and $f_2(M^{'}) \oplus f_2(M^{'} \oplus \delta)$ reach a distinguished point.

2. Store all endpoints of the constructed chains and search for collisions between the two different types of chains, *i.e.* for two plaintexts M and $M^{'}$ search for $(F_{f_1}(M))^a = (F_{f_2}(M^{'}))^b$ where $a, b \geq 1$ is the number of iterations of the respective functions. (For the rest of this paper, to facilitate the reading, the use of a and b will be omitted from equations that represent collisions between chains.)

3. If a collision is found, recover the XOR of the two inputs $M \oplus M^{'}$ which is expected to be equal to:

   $$K_1 \oplus K \oplus K_2 \oplus K = (\alpha + \alpha^2)K$$

   and thus recover the key K.

Analysis of the attack. To find a collision between the two sets of endpoints, one constructed by using the function F_{f_1} and one constructed by using F_{f_2}, we need to construct two chains each of length 2^{64}. So, the total cost of the attack is 2^{64}. As said previously, this attack does not contradict the security claim of Chaskey. It is just an example which shows how to use the distinguished point technique to attack Chaskey in the single-user setting.

4.2 Key-recovery attack in the multi-user setting

To apply the attack of Fouque *et al.* we use the iteration functions f_s, F_{f_s} and F_π defined previously. The attack is described below:

1. In a set of L users, for every user U_i where $0 \leq i \leq (L-1)$, build a constant number of chains, starting from an arbitrary plaintext, using the function F_{f_s}. For every user, create some chains using the function $F_{f_1}(M) = f_1(M) \oplus f_1(M \oplus \delta) \oplus M$ until $f_1(M) \oplus f_1(M \oplus \delta)$ reaches a distinguished point and some chains using the function $F_{f_2}(M) = f_2(M) \oplus f_2(M \oplus \delta) \oplus M$ until $f_2(M) \oplus f_2(M \oplus \delta)$ reaches a distinguished poit.

2. Construct some chains for the unkeyed user starting from an arbitrary plaintext, by iterating the function F_π.

3. Store the endpoints and search for collisions between the users. In the case of Chaskey, we can have three types of collisions between the keyed users:
   - A collision between the two chains $F_{f_1}(M)$ and $F_{f_2}(M^{'})$ of the same user i and, in this case, we recover $K_1^{(i)} \oplus K^{(i)} \oplus K_2^{(i)} \oplus K^{(i)}$. As $K_1^{(i)} = \alpha K^{(i)}$ and $K_2^{(i)} = \alpha^2 K^{(i)}$, we have that $M \oplus M^{'} = (\alpha + \alpha^2)K^{(i)}$.
   - A collision between two similar chains of two different users i and j: $F_{f_1}^{(i)}(M) = F_{f_1}^{(j)}(M^{'})$ or $F_{f_2}^{(i)}(M) = F_{f_2}^{(j)}(M^{'})$. Then, we recover $K_1^{(i)} \oplus K^{(i)} \oplus K_1^{(j)} \oplus K^{(j)} = (1+\alpha)(K^{(i)} \oplus K^{(j)})$ or $K_2^{(i)} \oplus K^{(i)} \oplus K_2^{(j)} \oplus K^{(j)} = (1+\alpha^2)(K^{(i)} \oplus K^{(j)})$.
   - A collision between two different type of chains between two different users i and j (cross collision): $F_{f_1}^{(i)}(M) = F_{f_2}^{(j)}(M^{'})$. Then, we learn $K_1^{(i)} \oplus K^{(i)} \oplus K_2^{(j)} \oplus K^{(j)} = (1+\alpha)K^{(i)} \oplus (1+\alpha^2)K^{(j)}$.

   Also search for collisions between a chain of a keyed user i and the unkeyed user for whom we build chains by using the function F_π. A collision of this type may occur by using both F_{f_1} or F_{f_2}. From the first function, we learn $K_1^{(i)} \oplus K^{(i)}$ and from the second function $K_2^{(i)} \oplus K^{(i)}$. As a consequence, in both cases, we are able to recover $K^{(i)}$.

4. Build a graph where vertices represent the keyed and unkeyed users. More precisely, each user is represented by two vertices as for each user we use two different functions to build chains. Whenever a collision is obtained between two chains, add an edge between the corresponding vertices. This edge is labelled by the relation between the keys as explained in the previous step. An example of the graph can be seen in Figure 3.

5. From the key relations found before, resolve the system and recover the users keys. With only a single collision for the unkeyed user, we learn almost all keys of the giant component. If we also find a collision for the unkeyed user, then we are able to learn all keys of our giant component.

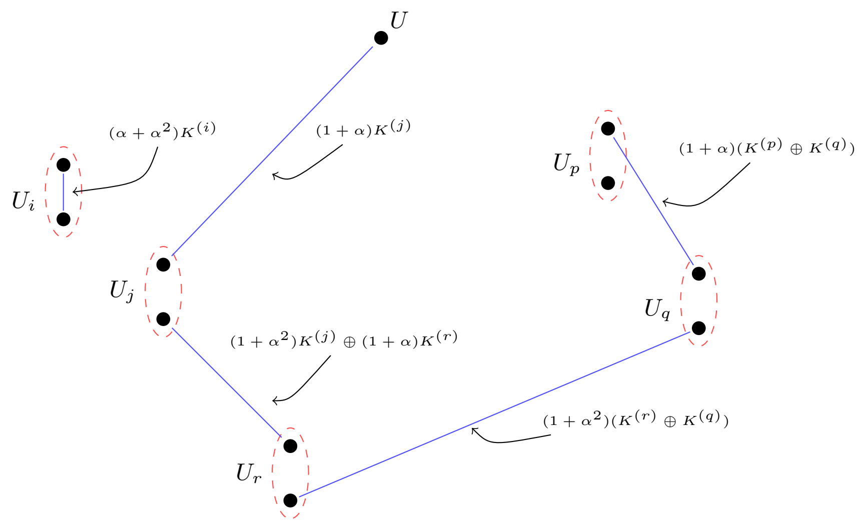
U
U_i
$(\alpha + \alpha^2)K^{(i)}$
$(1+\alpha)K^{(j)}$
U_p
$(1+\alpha)(K^{(p)} \oplus K^{(q)})$
U_j
U_q
$(1+\alpha^2)K^{(j)} \oplus (1+\alpha)K^{(r)}$
$(1+\alpha^2)(K^{(r)} \oplus K^{(q)})$
U_r

Fig. 3. Example of a giant component

Analysis of the attack. From [6] we know that in a group of $N^{1/3}$ users we expect to recover almost all keys by doing $c \cdot N^{1/3}$ queries per user (where c is a small arbitrary constant) and $N^{1/3}$ unkeyed queries. So, for Chaskey, we are able to recover almost all keys of a group of 2^{43} users by doing 2^{43} queries to the unkeyed user and 2^{43} queries per user.

Improvement. We show here that in the case of Chaskey, the use of δ can be eliminated. This technique can be used when we search for a collision between two similar chains of two different users.

If we use a full-block message x and its corresponding ciphertext y we observe that:

$$\begin{aligned}\alpha x \oplus (1+\alpha)y &= \alpha(x^{'} \oplus (1+\alpha)K) \oplus (1+\alpha)(y^{'} \oplus \alpha K)\\ &= \alpha x^{'} \oplus \cancel{\alpha(1+\alpha)K} \oplus (1+\alpha)y^{'} \oplus \cancel{\alpha(1+\alpha)K}\\ &= \alpha x^{'} \oplus (1+\alpha)y^{'}\end{aligned}$$

where $x, x^{'}, y$ and $y^{'}$ are represented in Figure 4.

$$\begin{tikzcd}
 & (1+\alpha)K \arrow[d] & & & & \alpha K \arrow[d] & \\
x \arrow[r] & \oplus \arrow[r] & x^{'} \arrow[r] & \boxed{\pi} \arrow[r] & y^{'} \arrow[r] & \oplus \arrow[r] & y
\end{tikzcd}$$

Fig. 4. The MAC algorithm Chaskey when single-block full messages are used (the subkey $K_1 = \alpha K$ is used).

As previously, our goal is to define a function and build chains by using this function. Here, we define the function $F_{f_1}(x)$ as follows:

$$F_{f_1}(x) = x \oplus \alpha x \oplus (1+\alpha) f_1(x) = (1+\alpha)(x \oplus f_1(x))$$

where $f_1(x) = K_1 \oplus \pi(x \oplus (K_1 \oplus K))$. We also assume that two plaintexts x_1 and x_2 satisfy $x_1 \oplus x_2 = (1+\alpha)(K^{(i)} \oplus K^{(j)})$.

Thus, for two users i and j for whom we detect a collision, we have that:

$$\begin{aligned}
f_1^{(i)}(x_1) \oplus f_1^{(j)}(x_2) &= \alpha(K^{(i)} \oplus K^{(j)}) \\
x_1 \oplus x_2 \oplus f_1^{(i)}(x_1) \oplus f_1^{(j)}(x_2) &= x_1 \oplus x_2 \oplus \alpha(K^{(i)} \oplus K^{(j)}) \\
(x_1 \oplus f_1^{(i)}(x_1)) \oplus (x_2 \oplus f_1^{(j)}(x_2)) &= K^{(i)} \oplus K^{(j)} \\
(1+\alpha)(x_1 \oplus f_1^{(i)}(x_1)) \oplus (1+\alpha)(x_2 \oplus f_1^{(j)}(x_2)) &= (1+\alpha)(K^{(i)} \oplus K^{(j)}) \\
F_{f_1}^{(i)} \oplus F_{f_1}^{(j)} &= (1+\alpha)(K^{(i)} \oplus K^{(j)})
\end{aligned}$$

and so we observe that chains constructed by F_{f_1} can become parallel and their constant difference would be equal to $(1+\alpha)(K^{(i)} \oplus K^{(j)})$.

A similar technique can be used when the last block is incomplete and so the key K_2 is used. Here, we observe that:

$$\begin{aligned}
\alpha^2 m \oplus (1+\alpha^2)\tau &= \alpha^2(m^{'} \oplus (1+\alpha^2)K) \oplus (1+\alpha^2)(\tau^{'} \oplus \alpha^2 K) \\
&= \alpha^2 m^{'} \oplus \cancel{\alpha^2(1+\alpha^2)K} \oplus (1+\alpha^2)\tau^{'} \oplus \cancel{\alpha^2(1+\alpha^2)K} \\
&= \alpha^2 m^{'} \oplus (1+\alpha^2)\tau^{'}.
\end{aligned}$$

Thus, in this case, we define the function F_{f_2} as follows:

$$F_{f_2}(x) = x \oplus \alpha^2 x \oplus (1+\alpha^2) f_2(x) = (1+\alpha^2)(x \oplus f_2(x))$$

where $f_2(x) = K_2 \oplus \pi(x \oplus (K_2 \oplus K))$. We also assume that two plaintexts x_1 and x_2 satisfy $x_1 \oplus x_2 = (1+\alpha^2)(K^{(i)} \oplus K^{(j)})$.
Equivalently, for two users i and j, we observe that:

$$F_{f_2}^{(i)} \oplus F_{f_2}^{(i)} = (1+\alpha^2)(K^{(i)} \oplus K^{(j)}).$$

So, chains constructed by F_{f_2} can also become parallel and their constant difference would be equal to $(1+\alpha^2)(K^{(i)} \oplus K^{(j)})$.

Thus, if we build our chains in the way presented here, we are able to have a small improvement and gain a factor of $\sqrt{2}$ on the calculation of our chains.

4.3 Variant of the previous attack with cross collisions

We show in this section that a variant of the previous attack is also possible. Indeed, we are able to apply a similar technique and we show that we can learn keys of two users when we detect one cross collision between them.

The attack works as follows:

1. For two users U_i and U_j in a set of L users, build a constant number of chains, starting from an arbitrary plaintext, using the function F_{f_s}, for $s \in \{1,2\}$. For each user, create some chains using the function $F_{f_1}(M) = f_1(M) \oplus f_1(M \oplus \delta) \oplus M$ and some chains using the function $F_{f_2}(M) = f_2(M) \oplus f_2(M \oplus \delta) \oplus M$. The construction of the chains stops when $f_1(M) \oplus f_1(M \oplus \delta)$ and $f_2(M) \oplus f_2(M \oplus \delta)$ reach a distinguished point.

2. For each chain, store the endpoints and search for a cross collision. A cross collision for users U_i and U_j and for two plaintexts M and M' is detected when $F_{f_1}^{(i)}(M) = F_{f_2}^{(j)}(M')$. This indicates that the XOR of the two inputs $M \oplus M'$ is expected to be equal to:
$$K_1^{(i)} \oplus K^{(i)} \oplus K_2^{(j)} \oplus K^{(j)} = (1+\alpha)K^{(i)} \oplus (1+\alpha^2)K^{(j)}.$$
3. If a cross collision is detected, recover also the XOR of the corresponding outputs, which is equal to:
$$K_1^{(i)} \oplus K_2^{(j)} = \alpha K^{(i)} \oplus \alpha^2 K^{(j)}.$$
4. Solve the system:
$$\begin{aligned}(1+\alpha)K^{(i)} \oplus (1+\alpha^2)K^{(j)} &= \Delta_1\\ \alpha K^{(i)} \oplus \alpha^2 K^{(j)} &= \Delta_2\end{aligned}$$
and thus recover $K^{(i)}$ and $K^{(j)}$.

Analysis of the attack. This attack uses similar techniques to the attack described before. However, the innovation here consists of the fact that we are able to recover keys of two users, in a smaller group than before, by finding only a cross collision between them. More specifically, to find a cross collision between two users, it suffices to have a group of $\sqrt[4]{N}$ users and perform $\sqrt[4]{N}$ queries per user. So, in the case of Chaskey, we are able to recover two keys in a group of 2^{32} by doing 2^{32} queries per user. However, if we want to recover the keys of all users, then we need a group of 2^{43} users as previously.

5 Conclusion

In this paper, we presented key-recovery attacks against the MAC algorithm Chaskey. All attacks are using algorithmic ideas for collision based attacks of Fouque *et al.* presented in [6]. They all work when using single-block messages in the single or multi-user setting.

First, we show how to use these techniques to form an attack in the classical single-user setting. By applying this attack, we are able to recover the key of the user by doing 2^{64} operations. However, this attack does not contradict the security claim of Chaskey. Next, we presented two attacks in the multi-user setting. The first one is able to recover almost all keys of a group of 2^{43} users by doing 2^{43} queries per user. We also show that we can improve this attack and gain a factor of $\sqrt{2}$. Finally, the second attack in the multi-user setting, is able to recover the keys of 2 users in a smaller group of 2^{32} by doing 2^{32} queries per user. We are able to achieve this new result by exploiting the use of two different keys for the last block of Chaskey.

References

1. Jean-Philippe Aumasson and Daniel J. Bernstein. Siphash: A fast short-input PRF. In *Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings*, pages 489–508, 2012.
2. Alex Biryukov and David Wagner. Advanced slide attacks. In *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, pages 589–606, 2000.
3. John Black and Phillip Rogaway. CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions. *J. Cryptology*, 18(2):111–131, 2005.
4. Orr Dunkelman, Nathan Keller, and Adi Shamir. Minimalism in cryptography: The even-mansour scheme revisited. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 336–354, 2012.
5. Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. In *Advances in Cryptology - ASIACRYPT '91, Fujiyoshida, Japan, November 11-14, 1991, Proceedings*, pages 210–224, 1991.
6. Pierre-Alain Fouque, Antoine Joux, and Chrysanthi Mavromati. Multi-user Collisions: Applications to Discrete Logarithm, Even-Mansour and PRINCE. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, pages 420–438, 2014.
7. Alfred Menezes. Another Look at Provable Security. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, page 8. Springer, 2012.
8. Nicky Mouha, Bart Mennink, Anthony Van Herrewege, Dai Watanabe, Bart Preneel, and Ingrid Verbauwhede. Chaskey: An efficient MAC algorithm for 32-bit microcontrollers. In *Selected Areas in Cryptography - SAC 2014 - 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers*, pages 306–323, 2014.