Chaskey: a Lightweight MAC Algorithm for Microcontrollers*

Nicky Mouha[1], Bart Mennink[1], Anthony Van Herrewege[1], Dai Watanabe[2], Bart Preneel[1], and Ingrid Verbauwhede[1]

[1]Department of Electrical Engineering, ESAT/COSIC, KU Leuven and iMinds, Belgium.
firstname.lastname@esat.kuleuven.be
[2]Yokohama Research Laboratory, Hitachi, Japan.
dai.watanabe.td@hitachi.com

Abstract. We propose Chaskey: a very efficient Message Authentication Code (MAC) algorithm for 32-bit microcontrollers. It is intended for applications that require 128-bit security, yet cannot implement standard MAC algorithms because of stringent requirements on speed, energy consumption, or code size. Chaskey is a permutation-based MAC algorithm that uses the Addition-Rotation-XOR (ARX) design methodology. We prove that Chaskey is secure in the standard model, based on the security of an underlying Even-Mansour block cipher. Chaskey is designed to perform well on a wide range of 32-bit microcontrollers. Our benchmarks show that on the ARM Cortex-M3/M4, our Chaskey implementation reaches a speed of 7.0 cycles/byte, compared to 89.4 cycles/byte for AES-128-CMAC. For the ARM Cortex-M0, our benchmark results give 16.9 cycles/byte and 136.5 cycles/byte for Chaskey and AES-128-CMAC respectively.

Keywords. Microcontroller, Message Authentication Code, Standard Model Security, Permutation-Based, ARX.

1 Introduction

Message Authentication Code (MAC) algorithms are one of the basic building blocks for cryptographic systems. A MAC algorithm processes a message m and a secret key K to generate a tag τ. It should be hard for an attacker to construct a forgery: that is, to generate a valid combination of (m, τ) without knowledge of the secret key K. Thereby, the MAC algorithm ensures the authenticity of the message m.

Over the years, a large variety of MAC algorithms have been proposed. Some of the most commonly used algorithms today are CMAC [33,41], HMAC [8,65], and UMAC [16]. CMAC is based on a block cipher, usually AES or Triple-DES, whereas HMAC uses a hash function such as MD5, SHA-1, or SHA-2, and UMAC is based on a universal hash function combined with a standard cryptographic primitive such as a block cipher or a hash function.

Unlike most other MAC algorithms, a nonce input is required for MAC algorithms based on universal hash functions [21,66]. This includes MAC algorithms such as UMAC [16], Poly1305-AES [11], and GMAC [34]. The nonce should not be reused, or this would lead to a forgery attack. Furthermore, Poly1305-AES and GMAC become insecure when tags are truncated [37]. We note that currently used MAC algorithms based on universal hash functions typically make use of multiplications. On several microcontrollers, the number of cycles required to execute an integer multiplication instruction is data-dependent, which makes the implementations potentially vulnerable to timing attacks [48].

For MAC algorithms that are based on hash functions, the block size is typically very large: for MD5, SHA-1, SHA-2, and the upcoming SHA-3 [12], messages are processed in blocks of

* An extended abstract of this paper appeared in Selected Areas in Cryptography - SAC 2014, Lecture Notes in Computer Science 8781, A. Joux and A. Youssef, Springer-Verlag, 2014.

at least 512 bits. For very short messages, this will result in a large overhead. But also for longer messages, it is generally undesirable for typical microcontrollers to process such large blocks. This is because many load and store operations are required to move data back and forth between the limited number of registers and the RAM, which significantly increases the time, energy, and code size of the MAC algorithm implementation.

A similar issue appears for block-cipher-based MAC algorithms, which typically use AES or Triple-DES. On typical microcontrollers, the key schedule of these block ciphers increases the register pressure: round keys must be either precomputed and stored in RAM, or computed on the fly. Furthermore, on 32-bit platforms, the S-box operations of AES and Triple-DES require extensive use of bit masking operations to implement the S-box operations, which again negatively impacts the speed of the implementation. Finally, we note that MAC algorithms based on reduced-round block ciphers such as ALPHA-MAC [24] and Pelican MAC [25] have been proposed, yet their performance gain is small for very short messages because a full-round block cipher is used for both initialization and finalization.

Chaskey

We present *Chaskey*, a permutation-based MAC algorithm that overcomes these issues. Chaskey takes a 128-bit key K and processes a message m in 128-bit blocks using a 128-bit permutation π. This permutation is based on the Addition-Rotation-XOR (ARX) design methodology. Its design is inspired by the permutation of SipHash [3], however with 32-bit instead of 64-bit words.

Chaskey has the following features:

- **Dedicated Design.** Chaskey is a dedicated design for 32-bit microcontroller architectures. The addition and XOR operations are performed on 32-bit words, and each of these operations requires only one instruction on these architectures.
- **Cross-Platform Versatility.** We took into account that certain microcontrollers do not support variable-length bit rotations and bit shifts. By choosing some rotation constants to be multiples of 8, these bit rotations are efficiently implemented by swapping 8-bit or 16-bit registers.
- **Efficient Implementation.** Benchmarks on an ARM Cortex-M4 show that Chaskey requires only 7.0 cycles/byte for long ($\geq$ 128 byte) messages, and 10.6 cycles/byte for short (16 byte) messages. It has been implemented in only 402 bytes of ROM. Results for the Cortex-M0 are very good as well: 16.9 cycles/byte for long messages, 21.3 cycles/byte for short ones, and 414 bytes of ROM for the implementation. There is, roughly speaking, a linear relation between the number of cycles and energy consumption [30]. We therefore expect Chaskey to be very energy efficient as well.
- **Resistance Against Timing Attacks.** On all microcontroller architectures that we are aware of, every instruction of Chaskey takes a constant time to execute. The total number of cycles depends only on the message length. Therefore, Chaskey is inherently secure against timing attacks.
- **Key Agility.** Chaskey does not have a key schedule, as keys are simply XORed into the state. Updating the key in Chaskey requires generating a new uniformly random 128-bit key, and only two shifts and two conditional XORs on 128-bit words to generate two subkeys.
- **Tag Truncation.** Chaskey is robust under tag truncation. Unlike for example GMAC [37], the best attack on Chaskey with short tags is tag guessing. We recommend $|\tau| \geq 64$ for typical applications. Shorter tags may be used after careful analysis of the probability of occasionally accepting an inauthentic message.

– **Nonces are Optional.** Several MAC algorithms (including GMAC [34], VMAC [49], and Poly1305-AES [11]) require a nonce, and become completely insecure if this nonce is reused (see e.g. [43]). Chaskey does not require a nonce, and therefore avoids these issues altogether.
– **Provably Secure.** We prove that Chaskey is secure, based on the security of an Even-Mansour [35, 36] block cipher based on π, up to about $D = 2^{64}$ blocks of chosen plaintexts and $T = 2^{128}/D$ off-line block cipher evaluations.
– **Patent-Free.** We are unaware of any patents or patent applications related to Chaskey.

The name Chaskey is derived from Chasqui, also written as Chaski. Chasquis were fast runners that delivered messages in the Inca empire. They were of short stature, and could cover large distances through mountainous areas with little nutrition available to them [56].

2 Preliminaries

Table 1 summarizes the notation used in this paper. Throughout, n is both the key size and the block size. While the Chaskey algorithm is introduced for $n = 128$, we remark that our statements on the Chaskey mode of operation are independent of this specific choice of n.

We interchangeably consider an element a of $GF(2^n)$ as an n-bit string $a[n-1]a[n-2]\dots a[0]$ and as the polynomial $a(x) = a[n-1]x^{n-1} + a[n-2]x^{n-2} + \dots + a[0]$ with binary coefficients. Let $f(x)$ be an irreducible polynomial of degree n with binary coefficients. For $n = 128$, we choose $f(x) = x^{128} + x^7 + x^2 + x + 1$. Then to multiply two elements a and b, we represent them as two polynomials $a(x)$ and $b(x)$, and calculate $a(x)b(x) \bmod f(x)$. For example, we show how to multiply an element by x in Algorithm 1. Note that x corresponds to bit string $0^{126}10$, which is 2 in decimal notation.

When converting between bit strings and arrays of 32-bit words, we always use little endian byte ordering. Inside every byte, bit numbering starts with the least significant bit.

Table 1: Notation.

Notation	Description
$x \| y$	concatenation of bit strings x and y
$\|x\|$	length of bit string x
$x + y$	addition of x and y modulo 2^{32} (in text)
$x \boxplus y$	addition of x and y modulo 2^{32} (in figures)
$x \lll s$	rotation of x to the left by s positions
$x \ll s$	shift of x to the left by s positions
$x \oplus y$	bitwise exclusive OR (XOR) of x and y
$\Delta^{\oplus} x$	XOR difference of x and x': $\Delta x = x \oplus x'$
0^a	bit string consisting of a times 0
$\mathsf{right}_t(a)$	select the t least significant bits of a
$x[i]$	bit selection: bit at position i of word x, where $i = 0$ is the least significant bit

3 Specification of Chaskey

3.1 Mode of Operation

Chaskey uses an n-bit key K to process a message m of arbitrary size into a tag τ of $t \leq n$ bits. For every key K, two subkeys K_1, K_2 are generated as shown in Algorithm 2.

The message m is split into ℓ blocks $m_1, m_2, \ldots, m_\ell$ of n bits each, except for the last block m_ℓ which may be incomplete. We define that an empty message $m = \varnothing$ consists of one empty block: $|m_1| = 0$. An n-bit permutation π then iterates over the message, as specified in Algorithm 3 and illustrated in Fig. 1.

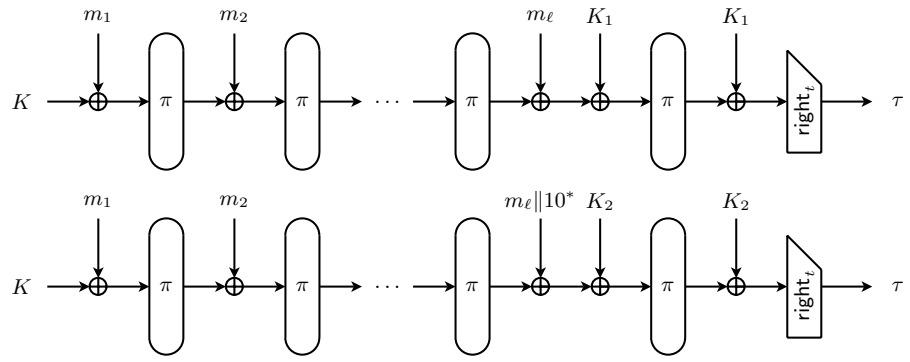

Fig. 1: The Chaskey mode of operation when $|m_\ell| = n$ (top), and when $0 \leq |m_\ell| < n$ (bottom). The round function of permutation π is shown in Fig. 3, the subkeys K_1 and K_2 are generated according to Algorithm 2, and $m_\ell\|10^*$ is shorthand for $m_\ell\|10^{n-|m_\ell|-1}$.

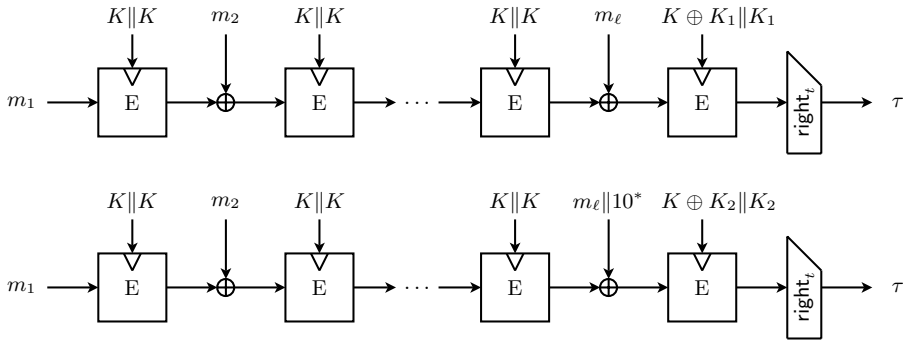

Fig. 2: The Chaskey-B mode of operation when $|m_\ell| = n$ (top), and $0 \leq |m_\ell| < n$ (bottom). Chaskey-B is an alternative description of Chaskey based on an Even-Mansour block cipher. The subkeys K_1 and K_2 are generated according to Algorithm 2, and $m_\ell\|10^*$ is shorthand for $m_\ell\|10^{n-|m_\ell|-1}$.

An alternative description of Chaskey based on an Even-Mansour [35,36] block cipher E with $2n$-bit key and n-bit block size is given in Algorithm 4, and illustrated in Fig. 2. This block-cipher-based description is equivalent to Chaskey once we define E using π as $E_{X\|Y}(m) = \pi(m \oplus X) \oplus Y$. The purpose of this block-cipher-based alternative is to reduce the security of Chaskey to the security of the underlying block cipher E. A security proof will be given in Sect. 5. This security proof views Chaskey-B as a variant of FCBC by Black and Rogaway [17,18], shown in Algorithm 5.

From this block-cipher-based description, it can be seen that Chaskey is similar to the three-key MAC constructions proposed by Black and Rogaway [17, 18]. Their constructions are variants of CBC-MAC [1, 40] that are secure for variable-length messages and avoid padding for messages of an integer number of blocks. As in CMAC [33, 41], our algorithm requires only

one n-bit key, from which two n-bit subkeys are generated. However, unlike CMAC, Chaskey does not require any block cipher calls to generate these two subkeys, only two shifts and two conditional XORs on 128-bit words.

Chaskey also differs from the CBC-MAC variants in literature because its underlying block cipher uses an Even-Mansour construction and as it uses the same subkey twice in the last two subkey XORs: before and after the last permutation call. Therefore, it is possible that this subkey (or part thereof) can remain inside the registers of the microcontroller. This reduces the number of load and store operations, which are very expensive on typical microcontrollers.

Every key K must be chosen independently and uniformly at random from the entire key space. To avoid attacks with a practical complexity of off-line permutation evaluations, as will be explained in Sect. 6.1, we restrict the total number of blocks to be authenticated under the same key K to at most 2^{48}. This corresponds to refreshing the key after at most 4 petabytes of data. To avoid tag guessing attacks, we recommend that the tag size $|\tau| \geq 64$. Changing $|\tau|$ always requires selecting a new key K uniformly at random.

Algorithm 1 TimesTwo

procedure TimesTwo(a)
  if $a[127] = 0$ **then**
    return $(a \ll 1) \oplus 0^{128}$
  else
    return $(a \ll 1) \oplus 0^{120}10000111$

Algorithm 2 SubKeys

procedure SubKeys(K)
  $K_1 \leftarrow \text{TimesTwo}(K)$
  $K_2 \leftarrow \text{TimesTwo}(K_1)$
  return (K_1, K_2)

Algorithm 3 Chaskey

procedure Chaskey$^{\pi}(K, m)$
  $(K_1, K_2) \leftarrow \text{SubKeys}(K)$
  $m_1 \| \ldots \| m_\ell \leftarrow m$
  $h_1 \leftarrow K$
  for $i = 1, \ldots, \ell - 1$ **do**
    $h_{i+1} \leftarrow \pi(h_i \oplus m_i)$
  if $|m_\ell| = n$ **then**
    $L \leftarrow K_1$
  else
    $m_\ell \leftarrow m_\ell \| 10^{n-|m_\ell|-1}$
    $L \leftarrow K_2$
  $h_{\ell+1} \leftarrow \pi(h_\ell \oplus m_\ell \oplus L) \oplus L$
  return $\tau \leftarrow \mathsf{right}_t(h_{\ell+1})$

Algorithm 4 Chaskey-B

procedure Chaskey-B$^{E}(K, m)$
  $(K_1, K_2) \leftarrow \text{SubKeys}(K)$
  $m_1 \| \ldots \| m_\ell \leftarrow m$
  $h_1 \leftarrow 0^n$
  for $i = 1, \ldots, \ell - 1$ **do**
    $h_{i+1} \leftarrow E_{K \| K}(h_i \oplus m_i)$
  if $|m_\ell| = n$ **then**
    $L \leftarrow K_1$
  else
    $m_\ell \leftarrow m_\ell \| 10^{n-|m_\ell|-1}$
    $L \leftarrow K_2$
  $h_{\ell+1} \leftarrow E_{K \oplus L \| L}(h_\ell \oplus m_\ell)$
  return $\tau \leftarrow \mathsf{right}_t(h_{\ell+1})$

Algorithm 5 FCBC [17, 18]

procedure FCBC($(p_1, p_2, p_3), m$)

  $m_1 \| \ldots \| m_\ell \leftarrow m$
  $h_1 \leftarrow 0^n$
  for $i = 1, \ldots, \ell - 1$ **do**
    $h_{i+1} \leftarrow p_1(h_i \oplus m_i)$
  if $|m_\ell| = n$ **then**
    $q \leftarrow p_2$
  else
    $m_\ell \leftarrow m_\ell \| 10^{n-|m_\ell|-1}$
    $q \leftarrow p_3$
  $h_{\ell+1} \leftarrow q(h_\ell \oplus m_\ell)$
  return $\tau \leftarrow h_{\ell+1}$

3.2 Permutation π

The permutation π is built using three operations: addition modulo 2^{32}, bit rotations, and XOR (ARX). The structure is the same as that of SipHash [3], but with 32-bit instead of 64-bit words and different rotation constants. Although SipHash has been proposed only very recently, it has found its way into several widely used software packages. For example, SipHash is used inside the hash table implementations of FreeBSD, Python, Perl, and Ruby [4]. Both Chaskey and SipHash use the 2-input MIX operation of Skein [38], one of the finalists of the SHA-3 competition [57].

In Chaskey, the permutation π consists of eight applications of a round function. This round function is specified in Fig. 3.

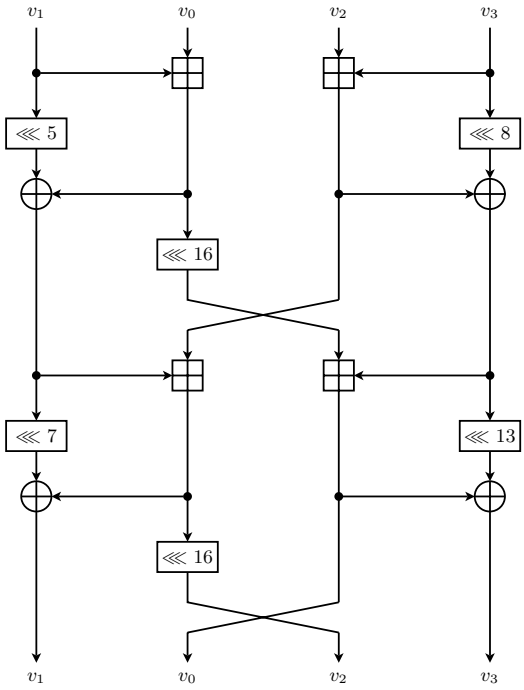

Fig. 3: A round of the Chaskey permutation π, defined as: $v_0\|v_1\|v_2\|v_3 \leftarrow \pi(v_0\|v_1\|v_2\|v_3)$. We intentionally swapped v_0 and v_1, as this reduces the number of crossing lines in the figure.

Although we are confident that eight rounds is enough for a secure construction, we recommend that implementers include the 16-round variant Chaskey-LTS (*long term security*) as a fallback in case of cryptanalytical breakthroughs. Chaskey-LTS consumes roughly twice the number of cycles and thus twice the amount of energy as Chaskey, but is still much faster than AES-CMAC. As only the number of rounds is different, it is possible to implement both Chaskey and Chaskey-LTS with negligible overhead in code size.

Note that half of the rotation constants of π are chosen to be multiples of eight. This is because a variety of microcontrollers do not support rotations and shifts over arbitrary amounts, e.g. the Renesas H8/300 CPU supports only one-bit rotations and shifts, the Renesas H8/2000 supports one-bit and two-bit rotations and shifts, and Microchip's 8-bit microcontrollers (PIC10/12/16/18) support one-bit rotations. Due to our choice of constants, implementation on 8- and 16-bit microcontrollers will be more efficient than had these constants been chosen at random. They furthermore allow us to implement Chaskey efficiently on a wide range of 32-bit microcontrollers, yet we have found that they do not seem to make π weaker against cryptanalytical attacks.

4 Implementation Results

We implemented Chaskey on several microcontroller platforms. We provide implementation results on ARM Cortex-M0 and -M4 platforms, and compare these to AES-128-CMAC on the same platforms. All our implementations have been compiled with GNU Tools for ARM Embedded Processors version 4.7.3 20121207. The Cortex-M0 benchmarks are executed on an STMicroelectronics STM32F030R8 microcontroller, the Cortex-M4 ones on an STM32F401RE.

We compare the results for our Chaskey implementation with what is, to the best of our knowledge, the fastest available AES implementation for the ARM Cortex-M series: SharkSSL [60,

61]. Since no AES-128-CMAC benchmarks are available for this implementation, we instead compare with AES-128-ECB, which is guaranteed to be at least as fast and small as AES-128-CMAC. Note that we list SharkSSL results for the Cortex-M3, since Cortex-M4 results are not available. However, the architecture of both microcontrollers is extremely similar, and thus results are expected to be the same.

Results for the various implementations are shown in Table 2. In all of our own benchmarks, round keys are precomputed, and time required to do so is not included in the listed numbers.

Table 2 shows that Chaskey compares favorably to AES-based MAC functions. On the ARM Cortex-M0 and -M4, Chaskey reaches 16.9 cycles/byte and 7.0 cycles/byte respectively. This is to be compared to 112.7 cycles/byte and 66.7 cycles/byte on the Cortex-M0 and -M3 respectively, using the AES-128-ECB implementation from SharkSSL [60, 61]. Furthermore, Chaskey can be implemented in 402 bytes and 414 bytes of ROM on respectively the Cortex-M0 and -M4. The SharkSSL AES-128-ECB implementation on the other hand, requires 4398 bytes and 3922 bytes for respectively a Cortex-M0 and -M3 implementation.

Table 2: Benchmark results for Chaskey and AES-128-CMAC on Cortex-M0/M4. AES-128-CMAC is implemented using AES code from the MAGEEC [53] framework. AES-128-ECB on Cortex-M0/M3 is based on figures from SharkSSL [60, 61]. Note that compiling with speed optimization flags does not always result in the fastest implementation.

Microcontroller	**Algorithm**	**Data [byte]**	**gcc flags**	**ROM size [byte]**	**Speed [cycles/byte]**
		Speed optimized			
Cortex-M0	AES-128-ECB (SharkSSL)	n/a	n/a	8 380	124.4
	AES-128-CMAC	128	-O2	13 492	136.5
	Chaskey	16	-O2	1 308	21.3
		128	-O2	1 308	18.3
Cortex-M3/M4	AES-128-ECB (SharkSSL)	n/a	n/a	4 854	66.7
	AES-128-CMAC	128	-O2	28 524	105.0
	Chaskey	16	-O2	908	10.6
		128	-O2	908	7.0
		Size optimized			
Cortex-M0	AES-128-ECB (SharkSSL)	n/a	n/a	4 398	112.7
	AES-128-CMAC	128	-Os	11 664	140.0
	Chaskey	16	-Os	414	21.8
		128	-Os	414	16.9
Cortex-M3/M4	AES-128-ECB (SharkSSL)	n/a	n/a	3 922	86.1
	AES-128-CMAC	128	-Os	10 952	89.4
	Chaskey	16	-Os	402	16.1
		128	-Os	402	11.2

5 Proof of Security

We focus on the security of the Chaskey mode of operation. For this, we consider $n, t \in \mathbb{N}$ to be arbitrary values. The proof consists of two phases. Firstly, we will prove the security of Chaskey-B in the standard model, based on any E with $2n$-bit key and n-bit block size. This will be done in Sect. 5.1. Next, in Sect. 5.2 we will show how these results generalize to Chaskey, once we use $E_{X\|Y}(m) = \pi(m \oplus X) \oplus Y$ for $\pi \in \{0,1\}^n$. In this phase of the proof we will employ

the ideal permutation model. Denote by $\mathsf{block}(k,n)$ the set of all block ciphers with k-bit key and n-bit block size, and let $\mathsf{perm}(n)$ denote the set of all permutations on n bits. Note that for $E \in \mathsf{block}(k,n)$, we have $E_K \in \mathsf{perm}(n)$ for all $K \in \{0,1\}^k$. The definitions below follow Bellare et al. [9] and Iwata and Kurosawa [41,42].

MAC Security. Let $\mathcal{H} : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^t$ be a MAC function.

$$\mathbf{Adv}^{\mathsf{mac}}_{\mathcal{H}}(q,D,r) = \max_{\mathcal{A}} \Pr\left(\begin{array}{l} K \xleftarrow{\$} \mathcal{K}\,,\ (m,\tau) \xleftarrow{\$} \mathcal{A}^{\mathcal{H}_K}\ ; \\ \mathcal{H}_K(m) = \tau \text{ and } m \text{ never queried} \end{array}\right),$$

where the maximum is taken over all adversaries making at most q queries of total length at most D blocks and running in time r.

VIPRF Security. We can define the variable input-length pseudorandom function (VIPRF) security as follows. Let $\mathcal{H} : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^t$ be a MAC function. Denote by $\mathsf{rand}(*,t)$ the set of all functions from $\{0,1\}^*$ to $\{0,1\}^t$. By $R \xleftarrow{\$} \mathsf{rand}(*,t)$ we mean that we consider a function R that associates to each input string $m \in \{0,1\}^*$ a random element from $\{0,1\}^t$.

$$\mathbf{Adv}^{\mathsf{viprf}}_{\mathcal{H}}(q,D,r) = \max_{\mathcal{A}} \left|\Pr\left(K \xleftarrow{\$} \mathcal{K}\ ;\ \mathcal{A}^{\mathcal{H}_K} = 1\right) - \Pr\left(R \xleftarrow{\$} \mathsf{rand}(*,t)\ ;\ \mathcal{A}^{R} = 1\right)\right|,$$

where the maximum is taken over all adversaries making at most q queries of total length at most D blocks and running in time r.

3PRP Security. The strength of a block cipher E is conventionally expressed as the PRP (pseudorandom permutation) security. In Chaskey-B (see Algorithm 4) we use a block cipher $E \in \mathsf{block}(2k,n)$ on input of three different keys: $E_{K\|K}$, $E_{K\oplus K_1\|K_1}$, and $E_{K\oplus K_2\|K_2}$, where K_1, K_2 are generated as shown in Algorithm 2. As the keys (K,K_1,K_2) are dependent, so are the three different usages of E. As such, a slightly more involved security notion is needed, which we call 3PRP. For ease of presentation, the definition is adapted to the specific key generation and block cipher use mode of Chaskey.

$$\mathbf{Adv}^{\mathsf{3prp}}_{E}(D,r) = \max_{\mathcal{A}} \left|\begin{array}{r} \Pr\left(\begin{array}{r} K \xleftarrow{\$} \{0,1\}^k,\ (K_1,K_2) \leftarrow \mathrm{SubKeys}(K)\ ; \\ \mathcal{A}^{E_{K\|K},E_{K\oplus K_1\|K_1},E_{K\oplus K_2\|K_2}} = 1 \end{array}\right) - \\ \Pr\left(p_1,p_2,p_3 \xleftarrow{\$} \mathsf{perm}(n)\ ;\ \mathcal{A}^{p_1,p_2,p_3} = 1\right) \end{array}\right|,$$

where the maximum is taken over all adversaries making at most D queries and running in time r.

5.1 Security of Chaskey-B

We start with the security of Chaskey-B of Algorithm 4. Let $E \in \mathsf{block}(2n,n)$ be any block cipher.

Theorem 1. *Let* $K \xleftarrow{\$} \{0,1\}^n$ *and consider* Chaskey-B$^E_K : \{0,1\}^* \to \{0,1\}^t$. *Then,*

$$\mathbf{Adv}^{\mathsf{mac}}_{\text{Chaskey-B}}(q,D,r) \leq \frac{2D^2}{2^n} + \frac{1}{2^t} + \mathbf{Adv}^{\mathsf{3prp}}_{E}(D,r).$$

Proof. Consider any adversary $\mathcal{A}$ that makes q queries of total length at most D blocks and that runs in time r. Note that Chaskey-B^E evaluates E for three different (but related) keys: $K\|K$, $K\oplus K_1\|K_1$, and $K\oplus K_2\|K_2$, where $(K_1, K_2) \leftarrow \text{SubKeys}(K)$. Let $p_1, p_2, p_3 \xleftarrow{\$} \mathsf{perm}(n)$ be three random permutations. As a first step, we replace these three different types of evaluations of E by p_1, p_2, p_3, and for simplicity call the resulting scheme Chaskey-PRP$_{p_1,p_2,p_3}$. A basic hybrid argument shows that their MAC security bounds differ by at most $\mathbf{Adv}^{\mathsf{3prp}}_E(D, r)$. Formally,

$$\mathbf{Adv}^{\mathsf{mac}}_{\text{Chaskey-B}}(q, D, r) \le \mathbf{Adv}^{\mathsf{mac}}_{\text{Chaskey-PRP}}(q, D, r) + \mathbf{Adv}^{\mathsf{3prp}}_E(D, r),$$

and it remains to analyze the security of Chaskey-PRP$_{p_1,p_2,p_3} : \{0,1\}^* \to \{0,1\}^t$, which is keyed via $(p_1, p_2, p_3) \xleftarrow{\$} \mathsf{perm}(n)^3$. We simplify the analysis, by giving $\mathcal{A}$ unlimited computational power and bounding $\mathbf{Adv}^{\mathsf{mac}}_{\text{Chaskey-PRP}}(q, D, \infty)$.

We note that

$$\text{Chaskey-PRP}_{p_1,p_2,p_3} = \mathsf{right}_t(\text{FCBC}_{p_1,p_2,p_3}),$$

where FCBC$_{p_1,p_2,p_3} : \{0,1\}^* \to \{0,1\}^n$ is given in Algorithm 5. Let $R \xleftarrow{\$} \mathsf{rand}(*, n)$. By a hybrid argument,

$$\mathbf{Adv}^{\mathsf{mac}}_{\text{Chaskey-PRP}}(q, D, \infty) \le \mathbf{Adv}^{\mathsf{viprf}}_{\text{FCBC}}(q, D, \infty) + \mathbf{Adv}^{\mathsf{mac}}_{\mathsf{right}_t(R)}(q, D, \infty).$$

Iwata and Kurosawa [42, Lem. 5.1] proved[3] that $\mathbf{Adv}^{\mathsf{viprf}}_{\text{FCBC}}(q, D, \infty) \le 2D^2/2^n$, where FCBC is keyed via $(p_1, p_2, p_3) \xleftarrow{\$} \mathsf{perm}(n)^3$. The function $\mathsf{right}_t(R)$ is, naturally, MAC secure up to $1/2^t$. This completes the proof. □

5.2 Security of Chaskey

We generalize the result of Sect. 5.1 to Chaskey of Algorithm 3.

The permutation π underlying Chaskey is in fact a publicly available permutation. In order to provide a security proof for the mode of operation, we consider Chaskey with idealized π. Consequently, we consider the security definitions of MAC, VIPRF, and 3PRP security where, additionally, $\pi \xleftarrow{\$} \mathsf{perm}(n)$ is randomly drawn and the adversary has query access to this underlying permutation.

Theorem 2. *Let $K \xleftarrow{\$} \{0,1\}^n$, assume that $\pi \xleftarrow{\$} \mathsf{perm}(n)$, and consider* Chaskey$^{\pi}_K : \{0,1\}^* \to \{0,1\}^t$. *Then,*

$$\mathbf{Adv}^{\mathsf{mac}}_{\text{Chaskey}}(q, D, r) \le \frac{2D^2}{2^n} + \frac{1}{2^t} + \frac{D^2 + 2DT}{2^n},$$

where T is defined as r/r_π for r_π denoting the running time of one evaluation of π.

Proof. Recall that Chaskey is equivalent to Chaskey-B once we select block cipher $E_{K\|L}(m) = \pi(m \oplus K) \oplus L$ based on $\pi \in \mathsf{perm}(n)$. In Lem. 1, it is proven that $\mathbf{Adv}^{\mathsf{3prp}}_E(D, r) \le (D^2 + 2DT)/2^n$, where $\mathcal{A}$ has access to either $(E_{K\|K}, E_{K\oplus K_1\|K_1}, E_{K\oplus K_2\|K_2}, \pi)$ or (p_1, p_2, p_3, π), and where $T = r/r_\pi$ corresponds to the number of (forward or inverse) permutation evaluations $\mathcal{A}$ can make. Together with Thm. 1, this completes the proof of Thm. 2. We remark that we can indeed apply Thm. 1 even though $\mathcal{A}$ has additional access to π: after the 3PRP swap, we consider $\mathbf{Adv}^{\mathsf{mac}}_{\text{Chaskey-PRP}}(q, D, \infty)$ where $\mathcal{A}$ has access to (Chaskey-PRP$_{p_1,p_2,p_3}, \pi$). As Chaskey-PRP$_{p_1,p_2,p_3}$ is independent of π, this permutation is irrelevant to $\mathcal{A}$ and we can ignore it. A similar reasoning holds for $\mathbf{Adv}^{\mathsf{viprf}}_{\text{FCBC}}(q, D, \infty)$ and $\mathbf{Adv}^{\mathsf{mac}}_{\mathsf{right}_t(R)}(q, D, \infty)$. □

[3] An earlier bound of $2.5D^2/2^n$ was derived by Black and Rogaway [17,18].

Lemma 1. *Under the notation of Thm. 2, we have* $\mathbf{Adv}^{\mathsf{3prp}}_{E}(D,r) \leq \dfrac{D^2+2DT}{2^n}$.

The proof is in part inspired by [41, Lem. 3] and in part by Patarin's H-coefficient technique [58]. We refer to Chen and Steinberger [23] for a detailed discussion of this technique. Note that the proof also generalizes the security analysis of the Even-Mansour block cipher [35, 36].

Proof. We consider an adversary $\mathcal{A}$ that has access to four oracles $(\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \mathcal{O}_4)$: in the real world, these are $(E_{K\|K}, E_{K\oplus K_1\|K_1}, E_{K\oplus K_2\|K_2}, \pi)$ for $\pi \xleftarrow{\$} \mathsf{perm}(n)$, $K \xleftarrow{\$} \{0,1\}^k$, and $(K_1, K_2) \leftarrow \mathrm{SubKeys}(K)$, and in the ideal world these are $(p_1, p_2, p_3, \pi) \xleftarrow{\$} \mathsf{perm}(n)^4$. The adversary can only make forward queries to O_1, O_2, and O_3, but has bidirectional access to O_4. We assume $\mathcal{A}$ is computationally unbounded, and without loss of generality that it is deterministic. It makes D_1 queries to its first oracle, D_2 to its second, and D_3 to its third, where $D_1 + D_2 + D_3 = D$, and T queries to $\mathcal{O}_4 = \pi$. We will be overly generous to the adversary: after it made all of its $D + T$ queries, but before it outputs its decision, we reveal the key K (in the real world) or send a randomly generated dummy key K (in the ideal world). We summarize the interaction of $\mathcal{A}$ with its oracles by a transcript $\tau = (K, \tau_1, \tau_2, \tau_3, \tau_4)$, where we denote by $\tau_j = \{(m_j^{(1)}, c_j^{(1)}), \ldots, (m_j^{(D_j)}, c_j^{(D_j)})\}$ the directionless list of queries to $\mathcal{O}_j$ for $j = 1, 2, 3$, and by $\tau_4 = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(T)}, y^{(T)})\}$ to $\mathcal{O}_4$. We assume the adversary never makes duplicate queries, hence $m_j^{(i)} \neq m_j^{(i')}$, $c_j^{(i)} \neq c_j^{(i')}$, $x^{(i)} \neq x^{(i')}$, and $y^{(i)} \neq y^{(i')}$ for all j, i, i'.

Denote by X (resp. Y) the probability distribution of transcripts in the real (resp. ideal) world, for fixed deterministic adversary $\mathcal{A}$. Say that a transcript τ is attainable if it can be obtained from interacting with (p_1, p_2, p_3, π), hence if $\Pr(Y = \tau) > 0$. The H-coefficient technique states the following, the proof of which we refer to [23].

Lemma 2 (H-coefficient Technique). *Consider a fixed deterministic adversary $\mathcal{A}$. let $\mathcal{T} = \mathcal{T}_{\mathrm{good}} \cup \mathcal{T}_{\mathrm{bad}}$ be a partition of the set of attainable transcripts. Let ε be such that for all $\tau \in \mathcal{T}_{\mathrm{good}}$*

$$\frac{\Pr(X = \tau)}{\Pr(Y = \tau)} \geq 1 - \varepsilon.$$

Then, $\mathbf{Adv}^{\mathsf{3prp}}_{E}(\mathcal{A}) \leq \varepsilon + \Pr(Y \in \mathcal{T}_{\mathrm{bad}})$.

Say that a transcript τ is *bad* if two different queries would lead to the same input or output to π, were $\mathcal{A}$ interacting with the real world. Formally, τ is bad if one of the following conditions is set:

$$\exists i, i' :\ m_1^{(i)} \oplus m_2^{(i')} = K_1 \qquad \vee\ c_1^{(i)} \oplus c_2^{(i')} = K \oplus K_1, \tag{1}$$
$$\exists i, i' :\ m_1^{(i)} \oplus m_3^{(i')} = K_2 \qquad \vee\ c_1^{(i)} \oplus c_3^{(i')} = K \oplus K_2, \tag{2}$$
$$\exists i, i' :\ m_2^{(i)} \oplus m_3^{(i')} = K_1 \oplus K_2 \ \vee\ c_2^{(i)} \oplus c_3^{(i')} = K_1 \oplus K_2, \tag{3}$$
$$\exists i, i' :\ m_1^{(i)} \oplus x^{(i')} = K \qquad \vee\ c_1^{(i)} \oplus y^{(i')} = K, \tag{4}$$
$$\exists i, i' :\ m_2^{(i)} \oplus x^{(i')} = K \oplus K_1 \ \vee\ c_2^{(i)} \oplus y^{(i')} = K_1, \tag{5}$$
$$\exists i, i' :\ m_3^{(i)} \oplus x^{(i')} = K \oplus K_2 \ \vee\ c_3^{(i)} \oplus y^{(i')} = K_2, \tag{6}$$

where $K_1 = \mathrm{TimesTwo}(K)$ and $K_2 = \mathrm{TimesTwo}(K_1)$. A transcript that is not bad is called *good*.

Upper Bounding $\Pr(Y \in \mathcal{T}_{\mathbf{bad}})$. Our goal is to bound the event that a transcript τ in the ideal world satisfies (1)–(6). Note that $K \xleftarrow{\$} \{0,1\}^n$ is a dummy key generated independently

of the transcripts $(\tau_1, \tau_2, \tau_3, \tau_4)$. Hence using the fact that $K \to K \oplus K_1$, $K \to K \oplus K_2$, and $K \to K_1 \oplus K_2$ are bijections, there are at most $2D_1D_2$ possible keys that would satisfy (1), $2D_1T$ possible keys for (4), and similar bounds for (2), (3), (5), and (6). We find

$$\begin{aligned}\Pr\left(Y \in \mathcal{T}_{\text{bad}}\right) &\leq \frac{2D_1D_2 + 2D_1D_3 + 2D_2D_3 + 2D_1T + 2D_2T + 2D_3T}{2^n}, \\ &\leq \frac{D^2 + 2DT}{2^n}.\end{aligned}$$

Lower Bounding Ratio $\Pr\left(X = \tau\right) / \Pr\left(Y = \tau\right)$. Consider a good and attainable transcript $\tau \in \mathcal{T}_{\text{good}}$. Denote by $\Omega_X = 2^n \cdot 2^n!$ the set of all possible oracles in the real world and by $\mathsf{comp}_X(\tau) \subseteq \Omega_X$ the set of oracles in Ω_X compatible with transcript τ. Define $\Omega_Y = 2^n \cdot (2^n!)^4$ and $\mathsf{comp}_Y(\tau)$ similarly. The H-coefficient technique dictates:

$$\Pr\left(X = \tau\right) = \frac{|\mathsf{comp}_X(\tau)|}{|\Omega_X|}, \qquad \text{and} \qquad \Pr\left(Y = \tau\right) = \frac{|\mathsf{comp}_Y(\tau)|}{|\Omega_Y|}.$$

We start with $|\mathsf{comp}_X(\tau)|$. As $\tau \in \mathcal{T}_{\text{good}}$, there are no two queries in τ with the same input to or output of the underlying permutation. Consequently, any query tuple in τ fixes exactly one input-output pair of the underlying oracle. As τ consists of $D + T$ query tuples, the number of possible oracles in the real world equals $(2^n - D - T)!$. Similarly, the number of possible oracles in the ideal world equals $\prod_{j=1}^{3}(2^n - D_j)!(2^n - T)!$. We thus find

$$\begin{aligned}\Pr\left(X = \tau\right) &= \frac{(2^n - D - T)!}{2^n \cdot 2^n!}, \\ \Pr\left(Y = \tau\right) &= \frac{\prod_{j=1}^{3}(2^n - D_j)!(2^n - T)!}{2^n \cdot (2^n!)^4} \leq \frac{(2^n - D - T)!(2^n!)^3}{2^n \cdot (2^n!)^4}.\end{aligned}$$

Concluding, $\Pr\left(X = \tau\right) / \Pr\left(Y = \tau\right) \geq 1$. □

6 Cryptanalysis

6.1 Attack Setting

In this section, we give an overview of the cryptographic properties of the Chaskey permutation π, and the two-key Even-Mansour block cipher $E_{X\|Y}(m) = \pi(m \oplus X) \oplus Y$. Note even if π has structural weaknesses, the security proof of Sect. 5.1 guarantees that Chaskey remains secure as long as $E_{K\|K}$, $E_{K\oplus K_1\|K_1}$, and $E_{K\oplus K_2\|K_2}$ are secure Even-Mansour block ciphers that are indistinguishable from each other. In particular, attackers are restricted to the following setting:

Uniformly Random Key K. Every implementation of Chaskey should ensure that the n-bit key K is chosen uniformly at random from the entire key space. In this way, Chaskey completely avoids all attacks on $E_{K\|K}$ using weak keys [26], known keys [46] or related keys [10, 13, 14]. In a weak-key attack, the attacker knows that the key K is chosen from a smaller subset of the key space. The attacker controls the value of K in a known-key attack, which in the case of the Even-Mansour block cipher corresponds to an attack on the underlying permutation π. In a related-key attack, the attacker obtains encryptions under different keys, and will know (or even control) the relationship among these keys.

Data Complexity D Below $2^{n/2}$ Chosen Plaintexts. No encryption device is allowed to perform close to $2^{n/2}$ block cipher calls under the same key. This is because after about $2^{n/2}$ block cipher calls, an internal collision attack [59] becomes likely. The same restriction applies to all iterated MAC constructions with an n-bit state. We will now explain that the data complexity under the same key should be restricted further to avoid attacks with a practical time complexity.

Time Complexity T Below $2^n/D$ Block Cipher Evaluations. Even and Mansour [35, 36] proved that any attack on their construction requires about T block cipher evaluations and $2^n/D$ known plaintexts. Dunkelman et al. [32] described a key recovery attack on the Even-Mansour construction to show that this bound is tight. As they clarify, this tight bound holds for both single-key and two-key Even-Mansour. To avoid attacks with a practical time complexity, the specification restricts the total number of blocks under the same key K to at most 2^{48}. This limit assumes that performing about 2^{80} off-line permutation evaluations is impractical for the attacker. Implementations that require a higher security level should rekey more frequently. We note that the amortized cost of rekeying is usually negligible, and rekeying does not require additional cryptographic components if Chaskey is also used as a key derivation function (KDF) [22].

No Chosen Ciphertext Attacks. The attacker cannot make any decryption queries $E^{-1}_{K\|K}$, $E^{-1}_{K\oplus K_1\|K_1}$, or $E^{-1}_{K\oplus K_2\|K_2}$, for the simple reason that Chaskey implementations do not contain the decryption function, and the corresponding keys are secret.

Tag Guessing Has Probability $2^{-|\tau|}$. The probability of constructing a forgery by guessing the tag is $2^{-|\tau|}$. Guessing a tag correctly for Chaskey does not make additional forgeries easier. The specification recommends that $|\tau| \geq 64$, which ensures that the probability of guessing τ correctly after 2^{32} trials is less than one in a billion. If it is acceptable to occasionally accept an inauthentic message as authentic (e.g. in certain voice communication applications [37]), the use of shorter tags may be carefully considered.

Implementation Attacks. Chaskey is inherently secure against timing attacks, as its execution time depends only on the message length $|m|$, and not on the secret key K. However, a straightforward implementation of Chaskey provides no resistance against hardware side channel attacks, nor to fault attacks. Furthermore, note that if the internal state of Chaskey is recovered and $|\tau| = n$, it is easy to recover the secret key K from any (m, τ)-pair.

6.2 Cryptanalysis of the Block Cipher

We now proceed with our cryptanalysis results for the block ciphers $E_{K\|K}$, $E_{K\oplus K_1\|K_1}$, and $E_{K\oplus K_2\|K_2}$ using π as the underlying permutation.

Standard Differential Cryptanalysis. We searched for differential characteristics of $E_{K\|K}$ that are linear in $GF(2)$, which means the output difference of every addition is the XOR of the two input differences. This was done by formulating this problem as the search for low-weight codewords in a linear code [63].

The best found characteristics for $1, 2, \ldots, 8$ rounds are shown in Table 3. We show only the input and output differences; the linearity property can be used to find the internal differences.

We calculated the characteristic probability in two ways: by determining the probability of every addition using the Lipmaa-Moriai formula [52] and multiplying these probabilities, and by using Leurent's ARX Toolkit [50, 51] to obtain a more accurate estimate that takes certain dependencies between operations into account.

In Table 4, we give the differences after every round of the best found differential characteristic for eight rounds, which corresponds to the last characteristic in Table 3. It is interesting to note that this characteristic has what can be described as an *hourglass* structure: the differences are sparse in the middle of the characteristics (located only in the most significant bits), and gradually become denser towards the outer rounds. The same observation also holds for all other characteristics of Table 3.

In Table 3, probabilities below 2^{-128} indicate that a characteristic exists only with some probability. Although such characteristics are not usable in an attack, it is important to explore them from a design point of view. Table 3 shows that Even-Mansour block ciphers based on π have a very large security margin against even very advanced variants of differential cryptanalysis attacks, especially as the data complexity in any attack on Chaskey is limited to 2^{64}.

Note that it is possible that better (possibly non-linear) characteristics exist, or that the probability of a given characteristic is lower than the probability of the corresponding differential. However, we expect that these effects will not be significant enough to invalidate our security claim against differential cryptanalysis.

Table 3: Best found differential characteristics for $1, 2, \ldots, 8$ rounds of the permutation π. Only the input and output differences are shown. Each of these characteristics is linear, this property can be used to determine the internal differences. We calculate the characteristic probability in two ways: assuming independence of every operation and using the Lipmaa-Moriai formula, as well as by Leurent's ARX Toolkit for a more refined estimate.

# Rounds	$\Delta_{\text{in}}^{\oplus}(v_0, v_1, v_2, v_3) \rightarrow \Delta_{\text{out}}^{\oplus}(v_0, v_1, v_2, v_3)$	Lipmaa-Moriai	Leurent
1	(00000000, 00000000, 80000000, 00000000) $\rightarrow$ (80000000, 80000000, 00008000, 80001000)	1	1
2	(00008400, 00000400, 00000000, 00000000) $\rightarrow$ (80008080, 00000040, 00000000, 80109080)	2^{-4}	2^{-4}
3	(00000008, 00000008, 00008181, 00000081) $\rightarrow$ (80109080, 80009810, 90108000, 92008082)	2^{-16}	2^{-16}
4	(C0240100, 44202100, 0C200008, 0C200000) $\rightarrow$ (10409000, 00547800, 18400010, 12408210)	2^{-37}	2^{-37}
5	(C8226120, 4C224101, 084C6908, 0C046900) $\rightarrow$ (E8001014, 08912214, 22100080, EA120916)	2^{-73}	$2^{-73.1}$
6	(1AC8DA46, 73C0D20A, 9282B2A3, 02947AA1) $\rightarrow$ (6A00109B, 50B7698C, 60001286, 68037999)	2^{-133}	$2^{-132.8}$
7	(8C74CC70, 7F3690AE, 5403A321, D1852232) $\rightarrow$ (DBCD9AC0, 293EC4DB, 6B1F0803, B195C08B)	2^{-208}	$2^{-205.6}$
8	(90EA132B, 88490EDB, 45854D95, E6A41996) $\rightarrow$ (726DC8C0, 097D6D14, 23822459, 2C2329AF)	2^{-293}	$2^{-289.9}$

Truncated Differential Cryptanalysis. We used the same techniques that were applied to Salsa20 [5] to find truncated differentials for $E_{K\|K}$. More specifically, we introduced differences in the most significant bits of the inputs, and searched for statistical biases in the output bits. We found such biases for up to four rounds of the block cipher. For example, if in the plaintext

Table 4: Best found linear differential characteristic for 8 rounds of π. This is the characteristic given in the last row of Table 3. If we assume independence of every operation and use the Lipmaa-Moriai formula for every addition, we find a probability of 2^{-293}. Leurent's ARX toolkit can be used to refine this probability to $2^{-289.9}$. Note the *hourglass* structure: differences are sparse in the middle, and gradually become denser towards the outer rounds.

Round_i	$\Delta^{\oplus}v_0$	$\Delta^{\oplus}v_1$	$\Delta^{\oplus}v_2$	$\Delta^{\oplus}v_3$	$\Pr[\text{Round}_{i-1} \rightarrow \text{Round}_i]$
0	90EA132B	88490EDB	45854D95	E6A41996	
1	1AC8DA46	73C0D20A	9282B2A3	02947AA1	2^{-76}
2	0C200008	08200008	81008104	81000085	2^{-55}
3	00000000	00000000	80800000	00800000	2^{-15}
4	00000000	80000000	00008000	00000000	2^{-1}
5	00000000	80008850	80108000	10000000	2^{-4}
6	18400010	18C02200	02401001	08421212	2^{-19}
7	6A00109B	50B7698C	60001286	68037999	2^{-39}
8	726DC8C0	097D6D14	23822459	2C2329AF	2^{-84}

$\Delta^{\oplus}v_1[31]$ and $\Delta^{\oplus}v_2[31]$ are both 1, then we found experimentally that $\Delta^{\oplus}v_2[16]$ after four rounds has a bias of about $2^{-12.48}$ towards 0. We tried out all combinations of input differences in the most significant bits of the four input words, but did not find biases in any of the output bit differences after five rounds or more, when experimenting with sets of 2^{30} samples.

Meet-in-the-Middle Attacks. The idea behind a meet-in-the-middle attack is to separate the mathematical equations that describe a block cipher into two or more groups, in such a way that some variables do not appear in at least one of the groups of equations. After three rounds of π, full diffusion occurs: every input bit affects every output bit. Similarly, π^{-1} also reaches full diffusion after three rounds. As eight rounds of π consist of almost three full diffusions, meet-in-the-middle attacks should not be applicable to Even-Mansour block ciphers based on π.

Note that the attacker is not allowed to perform chosen-ciphertext attacks, which limits the power of advanced meet-in-the-middle attacks, using the splice-and-cut technique that was introduced for hash function cryptanalysis [2, 64] and subsequently applied to block ciphers as well [20, 67].

A further extension of splice-and-cut meet-in-the-middle attacks are biclique attacks [19, 45]. Most applications of bicliques offer only slight improvements over brute force attacks [62]. Although brute-force-like attacks provide insight into the security of ciphers in the absence of other shortcut attacks, they do not affect the practical security of the cipher.

Rotational Cryptanalysis. A randomly chosen key K ensures that the input of the permutation π when used in an Even-Mansour block cipher will (with very high probability) have an asymmetrical state, thereby preventing rotational attacks [44].

Slide Attacks. Because every round of π is identical, slide attacks [15] are applicable to π. However, in a slide attack, about $2^{n/2}$ plaintext-ciphertext pairs are required before a slid pair is found. Therefore, slide attacks have a data complexity that goes beyond our security bound, and do not pose a threat to π, nor to Even-Mansour block ciphers based on π.

Fixed Points. Because π contains only the modular addition, XOR, and bitwise rotation operations, the permutation has the following fixed point: $\pi(0^n) = 0^n$. Fixed-points are a type

of differerentiability attack [54]. When π is used inside the $E_{K\|K}$ block cipher, this fixed point corresponds to $E_{K\|K}(K) = K$. If K is chosen uniformly at random, this relationship only holds with probability 2^{-n} for any plaintext chosen by the attacker. Similar observations hold for $E_{K\oplus K_1\|K_1}$ and $E_{K\oplus K_2\|K_2}$. Although it may seem to be a bold move from a design point of view to allow that $E_{0^{2n}}(0^n) = 0^n$, we note that this property also holds for the stream cipher Trivium [27, 29] and the block cipher KATAN [28]. However, no attacks have been found that break the full version of these ciphers.

Dependency Between Key and Subkeys. As shown by Algorithm 1, the subkeys K_1 and K_2 are generated from the key K as $K_1 = xK$ and $K_2 = x^2K_1$. The proof of Sect. 5.1 requires that an attacker cannot distinguish $E_{K\|K}$, $E_{(x+1)K\|xK}$, and $E_{(x^2+1)K\|x^2K}$ from each other. In Sect. 5.2, we already proved that this assumption holds if the underlying permutation π is an ideal permutation. We now argue that even if the permutation π of Sect. 3.2 is used instead, an attacker cannot distinguish these three block ciphers. Because of the rotational relations between the key K and the subkeys K_1 and K_2, rotational cryptanalysis [44] seems to be a promising technique. However, the fact that $(x + 1)K$ and xK, as well as $(x^2 + 1)K$ and x^2K both differ by K, seems to effectively preclude rotational cryptanalysis to distinguish $E_{(x+1)K\|xK}$ or $E_{(x^2+1)K\|x^2K}$ from $E_{K\|K}$, or from each other. Furthermore, the security proof assumes that individual queries to the three aforementioned block ciphers are permitted, whereas an attacker can in practice only observe τ.

Other Attacks. We do not consider zero-sum attacks [6] and cube attacks [31] to be a threat for ARX ciphers, because the addition operation ensures that for every output bit, the polynomial expression in $GF(2)$ representing this bit in terms of its inputs will be of sufficiently high degree. Moreover, rebound attacks [55] are not known to be relevant to secret-key algorithms.

7 Conclusion

Chaskey is a permutation-based MAC algorithm, with at its core an ARX-based permutation π based on SipHash. Alternatively, Chaskey can also be interpreted as a block-cipher-based MAC algorithm based on an underlying Even-Mansour block cipher.

Inspired by the block-cipher-based CMAC, Chaskey avoids padding for messages of an integer number of blocks. Its subkey generation is even more efficient than CMAC, as it does not require any block cipher calls.

We proved that Chaskey is secure, based on the 3PRP-indistinguishability of three underlying Even-Mansour block ciphers. Assuming that the permutation π used in these Even-Mansour block ciphers is ideal, we proved that Chaskey is secure up to about $D = 2^{n/2}$ chosen plaintexts and about $T = 2^n/D$ queries to π or π^{-1}.

We remark, however, that the efficient permutation π designed for Chaskey shows properties that allow it to be distinguished from an ideal permutation. For example, it is easy to find a fixed point: $\pi(0^n) = 0^n$. Fortunately, this observation does not extend to an attack when this permutation is used inside an Even-Mansour block cipher, as finding this fixed point implies knowledge of the secret key.

Therefore, we explored the distinguishability of the three Even-Mansour block ciphers from a cryptanalysis point of view. After investigating a wide variety of currently known cryptanalysis attacks, we found no shortcut attacks resulting from using our proposed eight-round permutation π instead of an ideal permutation. We recommend, however, that implementers also support a 16-round Chaskey-LTS as a fallback in case of cryptanalytical breakthroughs.

Our benchmarks showed that Chaskey performs very well on ARM Cortex-M microcontrollers. We measured that our straightforward Chaskey implementations are between 7 to 15 times faster than AES-128-CMAC in speed-optimized implementations, and at about 10 times smaller in area optimized implementations. Because of the roughly linear relation between cycle count and energy consumption, Chaskey is therefore much more energy efficient as well. Although 32-bit microcontrollers were our main target platform, Chaskey is also expected to perform well on 8-bit and 16-bit platforms.

Acknowledgments. This work was supported in part by the Research Council KU Leuven: GOA TENSE (GOA/11/007) and OT/13/071. Nicky Mouha and Bart Mennink are Postdoctoral Fellows of the Research Foundation – Flanders (FWO).

References

1. Ambler, E.: Computer Data Authentication. FIPS PUB 113, National Institute of Standards and Technology (NIST) (May 1985), `http://csrc.nist.gov/publications/fips/fips113/fips113.html`
2. Aoki, K., Sasaki, Y.: Preimage Attacks on One-Block MD4, 63-Step MD5 and More. In: Avanzi et al. [7], pp. 103–119
3. Aumasson, J.P., Bernstein, D.J.: SipHash: A Fast Short-Input PRF. In: Galbraith, S.D., Nandi, M. (eds.) INDOCRYPT. LNCS, vol. 7668, pp. 489–508. Springer (2012)
4. Aumasson, J.P., Bernstein, D.J.: SipHash: a fast short-input PRF. `https://131002.net/siphash/` (March 2014)
5. Aumasson, J.P., Fischer, S., Khazaei, S., Meier, W., Rechberger, C.: New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba. In: Nyberg, K. (ed.) FSE. LNCS, vol. 5086, pp. 470–488. Springer (2008)
6. Aumasson, J.P., Meier, W.: Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. Presented at the rump session of Cryptographic Hardware and Embedded Systems - CHES 2009 (2009)
7. Avanzi, R.M., Keliher, L., Sica, F. (eds.): Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers, LNCS, vol. 5381. Springer (2009)
8. Bellare, M., Canetti, R., Krawczyk, H.: Keying Hash Functions for Message Authentication. In: Koblitz [47], pp. 1–15
9. Bellare, M., Kilian, J., Rogaway, P.: The Security of Cipher Block Chaining. In: Desmedt, Y. (ed.) CRYPTO. LNCS, vol. 839, pp. 341–358. Springer (1994)
10. Bellare, M., Kohno, T.: A Theoretical Treatment of Related-Key Attacks: RKA-PRPs, RKA-PRFs, and Applications. In: Biham, E. (ed.) EUROCRYPT. LNCS, vol. 2656, pp. 491–506. Springer (2003)
11. Bernstein, D.J.: The Poly1305-AES Message-Authentication Code. In: Gilbert and Handschuh [39], pp. 32–49
12. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: The Keccak SHA-3 submission. Submission to the NIST SHA-3 Competition (Round 3) (2011)
13. Biham, E.: New Types of Cryptoanalytic Attacks Using related Keys (Extended Abstract). In: Helleseth, T. (ed.) EUROCRYPT. LNCS, vol. 765, pp. 398–409. Springer (1993)
14. Biham, E.: New Types of Cryptanalytic Attacks Using Related Keys. J. Cryptology 7(4), 229–246 (1994)
15. Biryukov, A., Wagner, D.: Slide Attacks. In: Knudsen, L.R. (ed.) FSE. LNCS, vol. 1636, pp. 245–259. Springer (1999)
16. Black, J., Halevi, S., Krawczyk, H., Krovetz, T., Rogaway, P.: UMAC: Fast and Secure Message Authentication. In: Wiener, M.J. (ed.) CRYPTO. LNCS, vol. 1666, pp. 216–233. Springer (1999)
17. Black, J., Rogaway, P.: CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions. In: Bellare, M. (ed.) CRYPTO. LNCS, vol. 1880, pp. 197–215. Springer (2000)
18. Black, J., Rogaway, P.: CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions. J. Cryptology 18(2), 111–131 (2005)
19. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique Cryptanalysis of the Full AES. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT. LNCS, vol. 7073, pp. 344–371. Springer (2011)
20. Bogdanov, A., Rechberger, C.: A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) Selected Areas in Cryptography. LNCS, vol. 6544, pp. 229–240. Springer (2010)
21. Carter, J.L., Wegman, M.N.: Universal Classes of Hash Functions. J. Comput. Syst. Sci. 18(2), 143–154 (1979)

22. Chen, L.: Recommendation for Key Derivation Using Pseudorandom Functions (Revised). NIST special publication 800-108, National Institute of Standards and Technology (NIST) (October 2009), http://csrc.nist.gov/publications/nistpubs/800-108/sp800-108.pdf
23. Chen, S., Steinberger, J.: Tight Security Bounds for Key-Alternating Ciphers. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT. LNCS, vol. 8441. Springer (2014)
24. Daemen, J., Rijmen, V.: A New MAC Construction ALRED and a Specific Instance ALPHA-MAC. In: Gilbert and Handschuh [39], pp. 1–17
25. Daemen, J., Rijmen, V.: The Pelican MAC Function. Cryptology ePrint Archive, Report 2005/088 (2005), http://eprint.iacr.org/
26. Davies, D.W.: Some Regular Properties of the 'Data Encryption Standard' Algorithm. In: Chaum, D., Rivest, R.L., Sherman, A.T. (eds.) CRYPTO. pp. 89–96. Plenum Press, New York (1982)
27. De Cannière, C.: Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. In: Katsikas, S.K., Lopez, J., Backes, M., Gritzalis, S., Preneel, B. (eds.) ISC. LNCS, vol. 4176, pp. 171–186. Springer (2006)
28. De Cannière, C., Dunkelman, O., Knezevic, M.: KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In: Clavier, C., Gaj, K. (eds.) CHES. LNCS, vol. 5747, pp. 272–288. Springer (2009)
29. De Cannière, C., Preneel, B.: Trivium. In: Robshaw, M.J.B., Billet, O. (eds.) The eSTREAM Finalists, LNCS, vol. 4986, pp. 244–266. Springer (2008)
30. de Clercq, R., Uhsadel, L., Van Herrewege, A., Verbauwhede, I.: Ultra Low-Power implementation of ECC on the ARM Cortex-M0+. In: The 51st Annual Design Automation Conference 2014, DAC '14, San Francisco, CA, USA, June 1-5, 2014. pp. 1–6. ACM (2014)
31. Dinur, I., Shamir, A.: Cube Attacks on Tweakable Black Box Polynomials. In: Joux, A. (ed.) EUROCRYPT. LNCS, vol. 5479, pp. 278–299. Springer (2009)
32. Dunkelman, O., Keller, N., Shamir, A.: Minimalism in Cryptography: The Even-Mansour Scheme Revisited. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT. LNCS, vol. 7237, pp. 336–354. Springer (2012)
33. Dworkin, M.: Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. NIST special publication 800-38b, National Institute of Standards and Technology (NIST) (May 2005), http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf
34. Dworkin, M.: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. NIST special publication 800-38d, National Institute of Standards and Technology (NIST) (November 2007), http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf
35. Even, S., Mansour, Y.: A Construction of a Cioher From a Single Pseudorandom Permutation. In: Imai, H., Rivest, R.L., Matsumoto, T. (eds.) ASIACRYPT. LNCS, vol. 739, pp. 210–224. Springer (1991)
36. Even, S., Mansour, Y.: A Construction of a Cipher from a Single Pseudorandom Permutation. J. Cryptology 10(3), 151–162 (1997)
37. Ferguson, N.: Authentication weaknesses in GCM. Comments submitted to NIST Modes of Operation Process (May 2005)
38. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The Skein Hash Function Family. Submission to the NIST SHA-3 Competition (Round 3) (2010), http://www.skein-hash.info/sites/default/files/skein1.3.pdf
39. Gilbert, H., Handschuh, H. (eds.): Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers, LNCS, vol. 3557. Springer (2005)
40. ISO/IEC: Information Technology: Information Technology – Security Techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms Using a Block Cipher. ISO/IEC 9797-1:2011 (2011)
41. Iwata, T., Kurosawa, K.: OMAC: One-Key CBC MAC. In: Johansson, T. (ed.) FSE. LNCS, vol. 2887, pp. 129–153. Springer (2003)
42. Iwata, T., Kurosawa, K.: Stronger Security Bounds for OMAC, TMAC, and XCBC. In: Johansson, T., Maitra, S. (eds.) INDOCRYPT. LNCS, vol. 2904, pp. 402–415. Springer (2003)
43. Joux, A.: Authentication Failures in NIST version of GCM. Comments submitted to NIST Modes of Operation Process (June 2006)
44. Khovratovich, D., Nikolić, I.: Rotational Cryptanalysis of ARX. In: Hong, S., Iwata, T. (eds.) FSE. LNCS, vol. 6147, pp. 333–346. Springer (2010)
45. Khovratovich, D., Rechberger, C., Savelieva, A.: Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family. In: Canteaut, A. (ed.) FSE. LNCS, vol. 7549, pp. 244–263. Springer (2012)
46. Knudsen, L.R., Rijmen, V.: Known-Key Distinguishers for Some Block Ciphers. In: Kurosawa, K. (ed.) ASIACRYPT. LNCS, vol. 4833, pp. 315–324. Springer (2007)
47. Koblitz, N. (ed.): Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings, LNCS, vol. 1109. Springer (1996)
48. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz [47], pp. 104–113

49. Krovetz, T.: Message Authentication on 64-Bit Architectures. In: Biham, E., Youssef, A.M. (eds.) Selected Areas in Cryptography. LNCS, vol. 4356, pp. 327–341. Springer (2006)
50. Leurent, G.: Analysis of Differential Attacks in ARX Constructions. In: Wang, X., Sako, K. (eds.) ASIACRYPT. LNCS, vol. 7658, pp. 226–243. Springer (2012)
51. Leurent, G.: Construction of Differential Characteristics in ARX Designs Application to Skein. In: Canetti, R., Garay, J.A. (eds.) CRYPTO. LNCS, vol. 8042, pp. 241–258. Springer (2013)
52. Lipmaa, H., Moriai, S.: Efficient Algorithms for Computing Differential Properties of Addition. In: Matsui, M. (ed.) FSE. LNCS, vol. 2355, pp. 336–350. Springer (2001)
53. MAGEEC (MAchine Guided Energy Efficient Compilation): `http://mageec.org` (2014)
54. Maurer, U.M., Renner, R., Holenstein, C.: Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In: Naor, M. (ed.) TCC. LNCS, vol. 2951, pp. 21–39. Springer (2004)
55. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In: Dunkelman, O. (ed.) FSE. LNCS, vol. 5665, pp. 260–276. Springer (2009)
56. Mozans, H.J.: Along the Andes and Down the Amazon, vol. 2. D. Appleton and company (1911)
57. National Institute of Standards and Technology: Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. Federal Register 27(212), 62212–62220 (November 2007), `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`
58. Patarin, J.: The "Coefficients H" Technique. In: Avanzi et al. [7], pp. 328–345
59. Preneel, B., van Oorschot, P.C.: MDx-MAC and Building Fast MACs from Hash Functions. In: Coppersmith, D. (ed.) CRYPTO. LNCS, vol. 963, pp. 1–14. Springer (1995)
60. RealTimeLogic: SHARKSSL v2.3.3 Crypto Library Benchmarks with ARM Cortex-M0@24MHz + ARM GCC 4.5.1. `http://realtimelogic.com/products/sharkssl/Cortex-M0/` (2014)
61. RealTimeLogic: SHARKSSL/RAYCRYPTO v2.4 Crypto Library Benchmarks with ARM Cortex-M3@50MHz + IAR EWARM 6.40. `http://realtimelogic.com/products/sharkssl/Cortex-M3/` (2014)
62. Rechberger, C.: On Bruteforce-Like Cryptanalysis: New Meet-in-the-Middle Attacks in Symmetric Cryptanalysis. In: Kwon, T., Lee, M.K., Kwon, D. (eds.) ICISC. LNCS, vol. 7839, pp. 33–36. Springer (2012)
63. Rijmen, V., Oswald, E.: Update on SHA-1. In: Menezes, A. (ed.) CT-RSA. LNCS, vol. 3376, pp. 58–71. Springer (2005)
64. Sasaki, Y., Aoki, K.: Preimage Attacks on Step-Reduced MD5. In: Mu, Y., Susilo, W., Seberry, J. (eds.) ACISP. LNCS, vol. 5107, pp. 282–296. Springer (2008)
65. Turner, J.M.: The Keyed-Hash Message Authentication Code (HMAC). FIPS PUB 198-1, National Institute of Standards and Technology (NIST) (July 2008), `http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf`
66. Wegman, M.N., Carter, J.L.: New Hash Functions and Their Use in Authentication and Set Equality. J. Comput. Syst. Sci. 22(3), 265–279 (1981)
67. Wei, L., Rechberger, C., Guo, J., Wu, H., Wang, H., Ling, S.: Improved Meet-in-the-Middle Cryptanalysis of KTANTAN (Poster). In: Parampalli, U., Hawkes, P. (eds.) ACISP. LNCS, vol. 6812, pp. 433–438. Springer (2011)