Generic Attacks Against Beyond-Birthday-Bound MACs

Gaëtan Leurent, Mridul Nandi, Ferdinand Sibleyras

Generic Attacks against Beyond-Birthday-Bound MACs

Gaëtan Leurent[1], Mridul Nandi[2], and Ferdinand Sibleyras[1]

[1] Inria, France
{gaetan.leurent,ferdinand.sibleyras}@inria.fr
[2] Indian Statistical Institute, Kolkata
mridul.nandi@gmail.com

Abstract. In this work, we study the security of several recent MAC constructions with provable security beyond the birthday bound. We consider block-cipher based constructions with a double-block internal state, such as SUM-ECBC, PMAC+, 3kf9, GCM-SIV2, and some variants (LightMAC+, 1kPMAC+). All these MACs have a security proof up to $2^{2n/3}$ queries, but there are no known attacks with less than 2^n queries.
We describe a new cryptanalysis technique for double-block MACs based on finding quadruples of messages with four pairwise collisions in halves of the state. We show how to detect such quadruples in SUM-ECBC, PMAC+, 3kf9, GCM-SIV2 and their variants with $\mathcal{O}(2^{3n/4})$ queries, and how to build a forgery attack with the same query complexity. The time complexity of these attacks is above 2^n, but it shows that the schemes do not reach full security in the information theoretic model. Surprisingly, our attack on LightMAC+ also invalidates a recent security proof by Naito.
Moreover, we give a variant of the attack against SUM-ECBC and GCM-SIV2 with time and data complexity $\tilde{\mathcal{O}}(2^{6n/7})$. As far as we know, this is the first attack with complexity below 2^n against a deterministic beyond-birthday-bound secure MAC.
As a side result, we also give a birthday attack against 1kf9, a single-key variant of 3kf9 that was withdrawn due to issues with the proof.

Keywords: Modes of operation, Cryptanalysis, Message Authentication Codes, Beyond-Birthday-Bound security

1 Introduction

Message authentication codes (or MACs) ensure the authenticity of messages in the secret-key setting. They are a core element of real-world security protocols such as TLS, SSH, or IPSEC. A MAC takes a message (and optionally a nonce) and a secret key to generate a tag that is sent with the message. Traditionally, they are classified into three types: deterministic, nonce-based, and probabilistic.

Deterministic MAC designs are the most popular, with widely used constructions based on block-cipher (CBC-MAC [13,4], OMAC [18], PMAC [5], LightMAC [29], ...) and hash functions (HMAC [2], NMAC [2], NI-MAC [1], ...). However, there is a generic forgery attack against all deterministic iterated MACs, using collisions in

the internal state, due to Preneel and van Oorschot [37]. Therefore, these MACs only achieve security up to the birthday bound, *i.e.* when the number of queries by the adversary is bounded by $2^{n/2}$, with n the state size. This is equivalently called $n/2$-bit security.

One way to increase the security is to use a *nonce*, a unique value provided by the user (in practice, the nonce is usually a counter). This approach has been pioneered by Wegman and Carter [41] based on an earlier work by Gilbert *et al.* [15]. Later a few follow ups like `EDM` and `EWCDM` [7], and `Dual EDM` [30] have been proposed to achieve beyond birthday security.

Alternatively, a probabilistic MAC uses a random coin for the extra value, which is usually called a *salt*, and must be transmitted with the MAC. Probabilistic MACs have the advantage that they can stay secure when called with the same input twice, and don't require a state to keep the nonce unique. Some popular probabilistic MAC constructions are `XMACR` [3], `RMAC` [22] and `EHtM` [31]. In particular, `RMAC` and `EHtM` have security beyond the birthday bound.

However, deterministic MACs are easier to use in practice, and there has been an important research effort to build deterministic MAC with security beyond the birthday bound, using an internal state larger than the primitive size. In particular, several constructions use a $2n$-bit internal state so that collisions in the state are only expected after 2^n queries. Yasuda first proposed `SUM-ECBC` [42], a beyond birthday bound (BBB) secure deterministic MAC that achieves $2n/3$-bit security. However, this construction has rate $1/2$ and later Yasuda himself proposed one of the most popular BBB secure MAC `PMAC+` [43] achieving rate 1. Later several other constructions like `3kf9` [44], `LightMAC+` [33], `GCM-SIV2` [20], and single key `PMAC+` [9] have been proposed. Interestingly, all the above designs share a common structure: a double-block universal hash function outputs a $2n$-bit hash value (seen as two n-bit halves), and a finalization function generates the tag by XORing encrypted values of the two n-bit hash values. This structure has been called double-block-hash-then-sum, and it will be the focus of our paper.

More recently, variants of `PMAC+` based on tweakable block-cipher have also been proposed, such as `PMAC_TBC` [32], `PMACx` [27], `ZMAC` [21], and `ZMAC+`[28].

Our results. We focus on the security of deterministic block-cipher based MACs with security beyond the birthday bound and double-block hash construction. Several previous works have been focused on security proofs, showing that they are secure up to $2^{2n/3}$ queries [43,44,20,9,42,33]. For most of these constructions, the advantage of an adversary making q short queries is bounded by $\mathcal{O}(q^3/2^{2n})$. Recently, Naito [34] gave an improved security proof for `LighMAC+`, with advantage at most $\mathcal{O}(q_t^2 q_v/2^{2n})$, with q_t MAC queries and q_v verification queries. In particular, this would prove security up to 2^n when the adversary can only do a single verification query.

In this work, we take the opposite approach and look for generic attacks against these modes. We use a cryptanalysis technique that can be seen as a generalisation of the collision attack of Preneel and van Oorschot [37]. Instead of looking for a pair of messages so that the full state collides, we look for a quadruple of messages, which can be seen either as two pairs colliding on the first

Table 1. Summary of the security for studied modes and our main results. q is the number of queries, ℓ is maximum size of a query, σ is total number of processed blocks. The expected lower bound and attack complexity is in number of constant length queries ($\ell = \mathcal{O}(1)$). We use "U" for universal forgeries, and "E" for existential forgeries.

Mode	Provable security bounds		Attacks (this work)		
	Advantage	Queries	Queries	Time	Type
`SUM-ECBC` [42]	$\mathcal{O}(\frac{q^3\ell^3}{2^{2n}})$	$\Omega(2^{2n/3})$	$\mathcal{O}(2^{3n/4})$	$\tilde{\mathcal{O}}(2^{3n/2})$	U
			$\mathcal{O}(2^{6n/7})$	$\tilde{\mathcal{O}}(2^{6n/7})$	U
`GCM-SIV2` [20]	$\mathcal{O}(\frac{q^3\ell^2}{2^{2n}})$	$\Omega(2^{2n/3})$	$\mathcal{O}(2^{3n/4})$	$\tilde{\mathcal{O}}(2^{3n/2})$	U
			$\mathcal{O}(2^{6n/7})$	$\tilde{\mathcal{O}}(2^{6n/7})$	U
`PMAC+` [43]	$\mathcal{O}(\frac{q^3\ell^3}{2^{2n}})$	$\Omega(2^{2n/3})$	$\mathcal{O}(2^{3n/4})$	$\tilde{\mathcal{O}}(2^{3n/2})$	E
`LightMAC+` [33]	$\mathcal{O}(\frac{q^3}{2^{2n}})$	$\Omega(2^{2n/3})$	$\mathcal{O}(2^{3n/4})$	$\tilde{\mathcal{O}}(2^{3n/2})$	E
`1kPMAC+` [9]	$\mathcal{O}(\frac{\sigma}{2^n} + \frac{q\sigma^2}{2^{2n}})$	$\Omega(2^{2n/3})$	$\mathcal{O}(2^{3n/4})$	$\tilde{\mathcal{O}}(2^{3n/2})$	E
`3kf9` [44]	$\mathcal{O}(\frac{q^3\ell^3}{2^{2n}} + \frac{q\ell}{2^n})$	$\Omega(2^{2n/3})$	$\mathcal{O}(\sqrt[4]{n} \cdot 2^{3n/4})$	$\tilde{\mathcal{O}}(2^{5n/4})$	U
`1kf9` [8]	$\mathcal{O}(\frac{q\ell^2}{2^n} + \frac{q^3\ell^4}{2^{2n}} + \frac{q^4\ell^4}{2^{3n}} + \frac{q^4\ell^6}{2^{4n}})$	$\Omega(2^{2n/3})$	$\mathcal{O}(2^{n/2})$	$\tilde{\mathcal{O}}(2^{n/2})$	U

half of the state, or two pairs colliding on the second half. Since the finalization function combines the halves with a sum, we can detect such a quadruple because the corresponding MACs sum to zero, and can usually amplify this filtering. Moreover, when the message are well constructed, the relations defining the four collisions create a linear system of rank only three, so that we expect one good quadruple out of 2^{3n}. Therefore, we only need four lists of $2^{3n/4}$ queries, and we expect one good quadruple out of the 2^{3n} choices in the four lists.

Table 1 shows a summary of our main results and how they compare with their respective provable security claims. In particular, we have forgeries attacks with $\mathcal{O}(2^{3n/4})$ MAC queries against `SUM-ECBC`, `GCM-SIV2`, `PMAC+`, `LightMAC+`, `1kPMAC+`, and `3kf9`. As far as we know, these are the first attacks with less than 2^n queries against these constructions. Our attack against `LighMAC+` contradicts the recent security bound for `LighMAC+` [34], because we have an attack with $\mathcal{O}(2^{3n/4})$ MAC queries, and a single verification query. The other attacks do not contradict the security proofs, but they make an important step towards understanding the actually security of these modes: we now have a lower bound of $2^{2n/3}$ queries from the proofs, and an upper bound of $2^{3n/4}$ from our attacks.

The attacks have a complexity of $2^{3n/4}$ in the information theoretic model (the model used for most MAC security proofs), but we note that an attacker needs more than 2^n operations to create a forgery. However, we have found a variant of our attack against `SUM-ECBC` and `GCM-SIV2` with total complexity below 2^n, using $\mathcal{O}(2^{6n/7})$ queries and $\tilde{\mathcal{O}}(2^{6n/7})$ operations.

We have also found an attack with only $\mathcal{O}(2^{n/2})$ queries and $\tilde{\mathcal{O}}(2^{n/2})$ operations against `1kf9` [8], a single key variant of `3kf9` with claimed security up to $2^{2n/3}$ queries. `1kf9` has been withdrawn due to issues with its security proof, but no attack was known previously.

Related works. There has been extensive work on security proofs for modes of operations, with a recent focus on security beyond the birthday bound. An interesting example is the encryption mode `CENC` by Iwata [17]: the initial proof was only up to $2^{2n/3}$ queries, but a later proof showed that it actually remains secure close to 2^n queries [19]. Our results show that in the case of double-block-hash-then-sum MACs, the security is lower than n-bit security.

Similarly, the initial proof of the randomized MAC `EHtM` only gave security up to $2^{2n/3}$, but a later proof showed security up to $2^{3n/4}$ [11]. This result also includes a matching attack, using a technique similar to ours based on looking for quadruples. However in the case of `EHtM` the attacker can observe part of the state, which allows him to find a right quadruple in $\mathcal{O}(2^{3n/4})$ time and memory. In our case we can't observe the internal state at all, thus we need to use different tricks tailored to each construction in order to amplify the filtering and avoid the many false-positives. In particular, this significantly increases the time and memory complexity.

There has also been intensive work on generic attacks to complement the security proof results. After the generic collision attack of Preneel and van Oorschot [37], more advanced attacks against MACs have been described, with stronger outcomes than existential forgeries, starting with a key-recovery attack against the envelop MAC by the same authors [38]. In particular, a series of attacks against hash-based MACs [26,36,16,10] led to universal forgery attacks against long challenges, and key-recovery attacks when the hash function has an internal checksum (like the GOST family). Against `PMAC`, Lee *et al.* showed a universal forgery attack in 2006 [25]. Later, Fuhr, Leurent and Suder gave a key-recovery attack against the `PMAC` variant used in AEZv3 [14]. Issues with `GCM` authentication with truncated tags were also pointed out by Ferguson [12]. These attacks don't contradict the security proofs of the schemes, but they are important results to understand the security degradation after the birthday bound.

Organization of the paper. We first explain our attack technique using quadruples of messages in Section 2, and give three concrete attacks using this technique: an attack against `SUM-ECBC` and `GCM-SIV2` in Section 3, an attack against `PMAC+` and related constructions in Section 4, and an attack against `3kf9` in Section 5. Finally, we show a variant of the technique using special properties of the single-key constructions of [8,9] in Section 6.

Notations. We denote the concatenation of messages blocks x and y as $x \parallel y$. When x and y fit together in one block, we use $x|y$ to denote their concatenation. We use $L[i]$ to denote element i of list L, $x_{[i]}$ to denote bit i of x, and $x_{[i:j]}$ to denote bits i to $j-1$. Finally, we use a curly brace for systems of equations.

2 Generic Attack against double-block-hash MACs

We first explain our attacks in a generic way, and leave the specific details to later sections focused on concrete MAC constructions.

We consider MACs where the $2n$-bit internal state is divided in two n-bit parts, that we denote Σ and Θ, and the final MAC is computed as:

$$\mathrm{MAC}(M) = E\big(\Sigma(M)\big) \oplus E'\big(\Theta(M)\big),$$

where E and E' denote the block cipher with potentially different keys. The functions Σ and Θ can be seen as two n-bit universal hash functions computed on the message, hence the name double-block-hash-then-sum MAC.

Our attacks exploit the fact that the two halves are combined with a sum, where one side depends only on Σ, and the other side depends only on Θ. They do not seem applicable to constructions with more intricate finalization functions, such as `LightMAC+2` [33], or the tweakable block-cipher based constructions `PMAC_TBC` [32], `PMACx` [27], `ZMAC` [21], or `ZMAC+`[28].

2.1 Using Quadruples

Our strategy consists in looking for a quadruple of messages (X, Y, Z, T) such that pairs of values collide for one half of the state. More precisely, we look for quadruples satisfying a relation $\mathcal{R}(X, Y, Z, T)$ defined as:

$$\mathcal{R}(X, Y, Z, T) := \begin{cases} \Sigma(X) = \Sigma(Y) \\ \Theta(Y) = \Theta(Z) \\ \Sigma(Z) = \Sigma(T) \\ \Theta(T) = \Theta(X) \end{cases}$$

In particular, since the MAC is computed as $\mathrm{MAC}(M) = E\big(\Sigma(M)\big) \oplus E'\big(\Theta(M)\big)$, it follows that:

$$\mathcal{R}(X, Y, Z, T) \implies \mathrm{MAC}(X) \oplus \mathrm{MAC}(Y) \oplus \mathrm{MAC}(Z) \oplus \mathrm{MAC}(T) = 0. \qquad (1)$$

In addition, if the messages X, Y, Z, T are well constructed, the relation $\mathcal{R}$ reduces to a linear system of rank only three, *i.e.*

$$\big[\Sigma(X) = \Sigma(Y) \text{ and } \Theta(Y) = \Theta(Z) \text{ and } \Sigma(Z) = \Sigma(T)\big] \implies \Theta(T) = \Theta(X).$$

Therefore, we expect to find one quadruple satisfying the relation out of 2^{3n}, and we can construct 2^{3n} quadruples with just $4 \times 2^{3n/4}$ queries. This gives an attack with data complexity $\mathcal{O}(2^{3n/4})$.

In practice, we consider lists of $2^{3n/4}$ messages, generated with two message injection functions ϕ and ψ. These functions are different in every attack, but they mostly correspond to adding two distinct prefixes, as in the following example:

$$\begin{aligned} \phi(i) &= 0 \,\|\, i & \psi(i) &= 1 \,\|\, i \\ X = \phi(x) &= 0 \,\|\, x & Y = \psi(y) &= 1 \,\|\, y \\ Z = \phi(z) &= 0 \,\|\, z & T = \psi(t) &= 1 \,\|\, t, \end{aligned}$$

In particular, the pairs (X, Y), (Y, Z), (Z, T) and (T, X) that we consider always contain a message built with ϕ and message built with ψ. Therefore, we will have the required collisions in Σ or Θ if the difference introduced in the half-state by the second block cancels the difference found after processing the first block.

This type of attack has some similarities with a higher order differential attack. Indeed, in the easiest case (*e. g.* our attack against `SUM-ECBC`), the relation $\mathcal{R}$ can be written as $\mathcal{R}(x, y, z, t) \iff \left[x \oplus y = z \oplus t = \Delta_1 \text{ and } x \oplus t = y \oplus z = \Delta_3\right]$ for some secret values Δ_1 and Δ_3. This idea of looking for quadruples is also very similar to the attack on `EHtM` [11], but the full attack will turn out quite different. Indeed, in the case of `EHtM`, the attacker can observe the salt R which represent half of the $2n$-bit internal state. Here this would be the equivalent of observing $\Sigma(m)$ for all processed messages m. This is clearly not possible for the studied constructions and we need something more to discriminate and find a good quadruple that satisfies $\mathcal{R}$.

2.2 Detecting Quadruples: Generalized Birthday Algorithms

To finish the attack we usually need to locate one good quadruple. The relation $\mathrm{MAC}(X) \oplus \mathrm{MAC}(Y) \oplus \mathrm{MAC}(Z) \oplus \mathrm{MAC}(T) = 0$ in itself is too weak because we expect one quadruple out of 2^n to satisfy it randomly, but we can usually amplify the filtering using related quadruples that satisfy $\mathcal{R}$ simultaneously (the exact details depend on the MAC construction).

In most of our attacks, we can express the search for a quadruple as an instance of the 4-sum problem, and solve it using variants of Wagner's generalized birthday algorithm [40]. This reduces the time complexity of the attacks (compared to a naive search), and provides trade-offs between the query, memory and time complexities.

More precisely, our problem can be stated as follow:

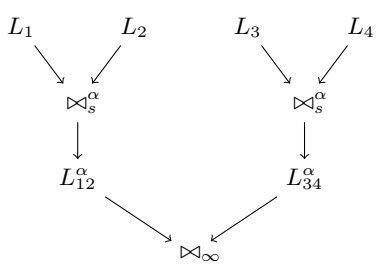
Definition 1 (4-sum problem). *Given four lists L_1, L_2, L_3, L_4 of 2^s elements, with on average 2^p quadruples $(x, y, z, t) \in L_1 \times L_2 \times L_3 \times L_4$ such that $x \oplus y \oplus z \oplus t = 0$, find one of them.*

Note that if the lists contain random n-bit words, we expect to have $2^p = 2^{4s-n}$ solutions, but in some of our instances there are more solutions because of the structure of the lists.

We denote the join operator as $\bowtie$; it computes the pairwise sum of two lists, and keeps the initial values attached to the sum. In addition, the join operator with filtering $\bowtie_t^\alpha$ only keeps values such that the t least significant bits of the sum agree with the value α:

$$A \bowtie B = \{(a \oplus b, a, b) \ : \ (a, b) \in A \times B\}$$
$$A \bowtie_t^\alpha B = \left\{(a \oplus b, a, b) \ : \ (a, b) \in A \times B,\ a_{[0:t]} \oplus b_{[0:t]} = \alpha\right\}$$

In particular, we have $\bowtie = \bowtie_0^0$. We also denote as $\bowtie_\infty$ the joint operator with filtering over the full input values. The filtered joint operator is the basis of Wagner's algorithm, and it can be computed in almost linear time by sorting the two input lists, and stepping through them simultaneously.



for $\alpha \in \{0,1\}^s$ **do**
  $L_{12}^\alpha \leftarrow L_1 \bowtie_s^\alpha L_2$
  $L_{34}^\alpha \leftarrow L_3 \bowtie_s^\alpha L_4$
  $L \leftarrow L_{12}^\alpha \bowtie_\infty L_{34}^\alpha$
  if $L \neq \varnothing$ **then**
    $(0,(x,y),(z,t)) \leftarrow L[0]$
    return (x,y,z,t)
  end if
end for
return $\perp$

Fig. 1. Generalized Birthday algorithm to find good quadruples.

Direct algorithm. While a naive algorithm for our 4-sum instances would take time 2^{4s} to examine all quadruples, there is a simple improvement with time and memory $\tilde{\mathcal{O}}(2^{2s})$. First, the attacker builds $L_{12} = L_1 \bowtie L_2$ and $L_{34} = L_3 \bowtie L_4$. Then, he looks for a collision between the first component of L_{12} and L_{34}. A collision directly yields a solution. This always finds a solution if it exists in $\tilde{\mathcal{O}}(2^{2s})$ operations but it also takes $\mathcal{O}(2^{2s})$ memory.

Memory efficient algorithm. We can reduce the memory complexity of the algorithm if we avoid constructing the full lists L_{12} and L_{34}. An algorithm with low memory complexity was first described by Chose *et al.* [6], but we use the description given by Wagner in the full version of [40].

Instead of building the full lists L_{12} and L_{34}, we filter values such that s least significant bits differ by some fixed value α. This reduces the expected size of the lists to only 2^s: $E[|L_{34}^\alpha|] = E[|L_{12}^\alpha|] = |L_1| \cdot |L_2|/2^s = 2^s$. If this algorithm is repeated for every s-bit value α, it will eventually find all solutions.

Actually, one run of the algorithm detects the solutions whose least significant bits of $x \oplus y$ are equal to α. If there are 2^p solutions in total, there is one such solution with probability 2^{p-s}, and this algorithm will find the first solution after trying 2^{s-p} values of α on average. Therefore, the expected time complexity of the algorithm given by Figure 1 is only $\tilde{\mathcal{O}}(2^{2s-p})$.

Related work. In a 2016 work, Nikolic and Sasaki [35] investigate the 4-sum where we need to find 4 different inputs x, y, z, t to a function f such that $f(x) \oplus f(y) \oplus f(z) \oplus f(t) = 0$. They also mention that their algorithm is adaptable to pairwise identical functions, *i. e.* $f(x) \oplus g(y) \oplus f(z) \oplus g(t) = 0$.

Most of our attacks can be written in this way; concretely, they are equivalent to instances of random functions with $3n$-bit outputs. In this setting our algorithm takes time $\tilde{\mathcal{O}}(2^{3n/2})$ and memory $\mathcal{O}(2^{3n/4})$, while Nikolic and Sasaki's work can reach $\tilde{\mathcal{O}}(2^{9n/8})$ time and $\mathcal{O}(2^{3n/4})$ memory. Unfortunately, their algorithm requires $\tilde{\mathcal{O}}(2^{9n/8})$ queries to the functions; this would translate to $\tilde{\mathcal{O}}(2^{9n/8})$ queries to the MAC, which is not interesting in our context.

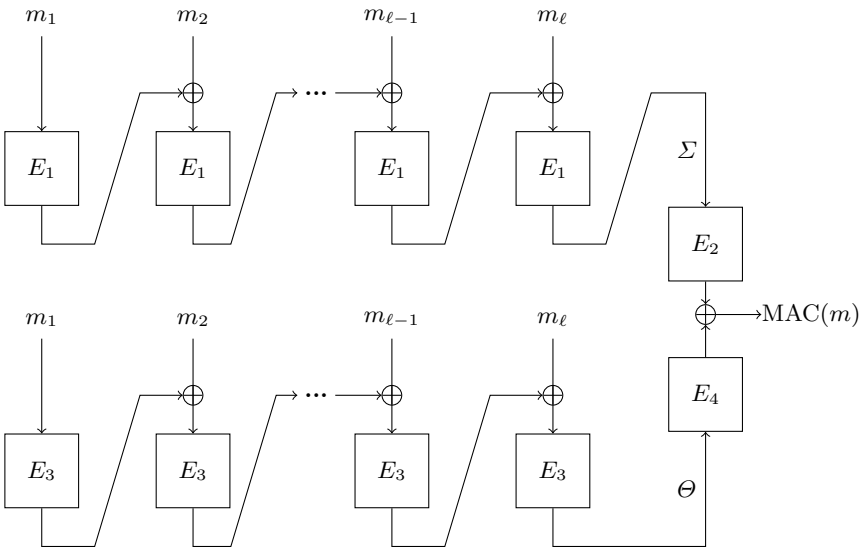


Fig. 2. Diagram for `SUM-ECBC` with a $\ell-$block message.

3 Attacking `SUM-ECBC`-like constructions

We start with attacks against `SUM-ECBC` [42] and `GCM-SIV2` [20]; while the constructions are quite different, they have a similar structure and the same attacks can be used in both cases. We give a universal forgery attack with $\mathcal{O}(2^{3n/4})$ queries and $\tilde{\mathcal{O}}(2^{3n/2})$ operations (using memory $\mathcal{O}(2^{3n/4})$), and a variant with total complexity below 2^n, with $\mathcal{O}(2^{6n/7})$ queries and $\tilde{\mathcal{O}}(2^{6n/7})$ operations.

3.1 Attacking `SUM-ECBC`

`SUM-ECBC` was designed by Yasuda in 2010 [42], inspired by MAC constructions summing two CBC-MACs in the ISO 9797-1 standard. The scheme uses a block cipher keyed with four independent keys, denoted as E_1, E_2, E_3, E_4. The message M is first padded with 10^* padding, and divided into n-bit blocks. In the following we ignore the padding and consider the padded message as the input: this makes our description easier, and any padded message whose last block is non-zero can be "un-padded" to generate a valid input message. The construction is defined as follows (see also Figure 2):

$$\begin{aligned} \Sigma(M) &= \sigma_\ell & \sigma_0 &= 0 \qquad \sigma_i = E_1(\sigma_{i-1} \oplus m_i) \\ \Theta(M) &= \theta_\ell & \theta_0 &= 0 \qquad \theta_i = E_3(\theta_{i-1} \oplus m_i) \\ \mathrm{MAC}(M) &= E_2(\Sigma(M)) \oplus E_4(\Theta(M)) \end{aligned}$$

Attack. Following the framework of Section 2, we consider quadruple of messages, built with two message injection functions:

$$\phi(i) = 0 \,\|\, i \qquad\qquad \psi(i) = 1 \,\|\, i$$

In particular, we have

$$\mathrm{MAC}(\phi(i)) = E_2\Big(\underbrace{E_1\big(i \oplus E_1(0)\big)}_{\Sigma_0(i)}\Big) \oplus E_4\Big(\underbrace{E_3\big(i \oplus E_3(0)\big)}_{\Theta_0(i)}\Big)$$
$$\mathrm{MAC}(\psi(i)) = E_2\Big(\underbrace{E_1\big(i \oplus E_1(1)\big)}_{\Sigma_1(i)}\Big) \oplus E_4\Big(\underbrace{E_3\big(i \oplus E_3(1)\big)}_{\Theta_1(i)}\Big)$$

Next, we build quadruples of messages X, Y, Z, T with

$$X = \phi(x) \qquad Y = \psi(y) \qquad Z = \phi(z) \qquad T = \psi(t),$$

and we look for a quadruple with partial state collisions for the underlying pairs, *i. e.* a quadruple following the relation:

$$\mathcal{R}(x,y,z,t) := \begin{cases} \Sigma_0(x) = \Sigma_1(y) \\ \Sigma_0(z) = \Sigma_1(t) \\ \Theta_0(z) = \Theta_1(y) \\ \Theta_0(x) = \Theta_1(t). \end{cases}$$

We have

$$\mathcal{R}(x,y,z,t) \Leftrightarrow \begin{cases} x \oplus E_1(0) = y \oplus E_1(1) \\ z \oplus E_3(0) = y \oplus E_3(1) \\ z \oplus E_1(0) = t \oplus E_1(1) \\ x \oplus E_3(0) = t \oplus E_3(1) \end{cases} \Leftrightarrow \begin{cases} x \oplus y \oplus z \oplus t = 0 \\ x \oplus y = E_1(0) \oplus E_1(1) \\ x \oplus t = E_3(0) \oplus E_3(1) \end{cases}$$

As promised in Section 2, $\mathcal{R}$ defines a $3n-$bit relation. We can easily observe when $x \oplus y \oplus z \oplus t = 0$, and we can also detect the relation on the sum of the MACs following Equation (1):

$$\mathcal{R}(x,y,z,t) \Rightarrow \mathrm{MAC}(\phi(x)) \oplus \mathrm{MAC}(\psi(y)) \oplus \mathrm{MAC}(\phi(z)) \oplus \mathrm{MAC}(\psi(t)) = 0$$

Moreover, we observe that $\mathcal{R}(x,y,z,t)$ is satisfied if and only if $\mathcal{R}(x \oplus c, y \oplus c, z \oplus c, t \oplus c)$ is satisfied for any constant c. We use this relation to build several quadruples that satisfy $\mathcal{R}$ simultaneously:

$$\mathcal{R}(x,y,z,t) \iff \forall c,\, \mathcal{R}(x \oplus c, y \oplus c, z \oplus c, t \oplus c) \tag{2}$$

This leads to an attack with $\mathcal{O}(2^{3n/4})$ queries: we consider four sets $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{T}$ of $2^{3n/4}$ values, and we look for a quadruple $(x,y,z,t) \in \mathcal{X} \times \mathcal{Y} \times \mathcal{Z} \times \mathcal{T}$ with:

$$\begin{cases} x \oplus y \oplus z \oplus t = 0 \\ \mathrm{MAC}(\phi(x)) \oplus \mathrm{MAC}(\psi(y)) \oplus \mathrm{MAC}(\phi(z)) \oplus \mathrm{MAC}(\psi(t)) = 0 \\ \mathrm{MAC}(\phi(x \oplus 1)) \oplus \mathrm{MAC}(\psi(y \oplus 1)) \oplus \mathrm{MAC}(\phi(z \oplus 1)) \oplus \mathrm{MAC}(\psi(t \oplus 1)) = 0. \end{cases} \tag{3}$$

Because we need a fair distribution of values $x \oplus y$ and $x \oplus t$ to find the good quadruple we build the sets as:

$$\mathcal{X} = \left\{x \in \{0,1\}^n \ : \ x_{[0:n/4]} = 0\right\} \qquad \mathcal{Y} = \left\{x \in \{0,1\}^n \ : \ x_{[n/4:n/2]} = 0\right\}$$
$$\mathcal{Z} = \left\{x \in \{0,1\}^n \ : \ x_{[n/2:3n/4]} = 0\right\} \qquad \mathcal{T} = \left\{x \in \{0,1\}^n \ : \ x_{[3n/4:n]} = 0\right\}$$

With this construction, there is exactly one quadruple $(x, y, z, t) \in \mathcal{X} \times \mathcal{Y} \times \mathcal{Z} \times \mathcal{T}$ that respects $\mathcal{R}$, given by:

$$x = v_1|w_2|u_3|0 \qquad y = w_1|v_2|0|u_4 \qquad z = u_1|0|v_3|w_4 \qquad t = 0|u_2|w_3|v_4,$$

where:

$$\begin{aligned} E_1(0) \oplus E_1(1) &=: u_1|u_2|u_3|u_4 \\ E_3(0) \oplus E_3(1) &=: v_1|v_2|v_3|v_4 \\ E_1(0) \oplus E_1(1) \oplus E_3(0) \oplus E_3(1) &=: w_1|w_2|w_3|w_4. \end{aligned}$$

We expect on average one random quadruple satisfying (3) (with 2^{3n} potential quadruples, and a $3n$-bit filtering), in addition to the quadruple satisfying $\mathcal{R}$. The correct quadruple can easily be checked with a few extra queries.

In practice, we use the generalized birthday algorithms of Section 2.2 in order to optimize the complexity of the attack. We consider four lists:

$$\begin{aligned} L_1 &= \{x \parallel \mathrm{MAC}(\phi(x)) \parallel \mathrm{MAC}(\phi(x \oplus 1)) \ : \ x \in \mathcal{X}\} \\ L_2 &= \{y \parallel \mathrm{MAC}(\psi(y)) \parallel \mathrm{MAC}(\psi(y \oplus 1)) \ : \ y \in \mathcal{Y}\} \\ L_3 &= \{z \parallel \mathrm{MAC}(\phi(z)) \parallel \mathrm{MAC}(\phi(z \oplus 1)) \ : \ z \in \mathcal{Z}\} \\ L_4 &= \{t \parallel \mathrm{MAC}(\psi(t)) \parallel \mathrm{MAC}(\psi(t \oplus 1)) \ : \ t \in \mathcal{T}\} \end{aligned}$$

Notice that we can build those lists with $5 \cdot 2^{3n/4}$ queries as, by construction, for any element i of $\mathcal{Y}, \mathcal{Z}, \mathcal{T}$ the element $(i \oplus 1)$ also belongs to $\mathcal{Y}, \mathcal{Z}, \mathcal{T}$, respectively. We use the algorithm of Section 2.2 to find $(x, y, z, t) \in \mathcal{X} \times \mathcal{Y} \times \mathcal{Z} \times \mathcal{T}$ such that $L_1[x] \oplus L_2[y] \oplus L_3[z] \oplus L_4[t] = 0$ with $\tilde{\mathcal{O}}(2^{3n/2})$ operations, using a memory of size $\mathcal{O}(2^{3n/4})$. After finding a collision, we verify that it is not a false positive by testing the relation for another value c. As there are on average $\mathcal{O}(1)$ random quadruples the attack is indeed using a total of $5 \cdot 2^{3n/4} + \mathcal{O}(1) = \mathcal{O}(2^{3n/4})$ queries.

Universal Forgeries. This attack can be extended to a universal forgery. Indeed, the fixed prefix 0 and 1 can be replaced by v and $v \oplus 1$ for any block v, and when we identify a right quadruple (x, y, z, t) we deduce the value $\Delta_1 = E_1(v) \oplus E_1(v \oplus 1)$ and $\Delta_3 = E_3(v) \oplus E_3(v \oplus 1)$. There is also a length extension property: if (x, y, z, t) is a right quadruple, then $\mathrm{MAC}(v \parallel x \parallel s) \oplus \mathrm{MAC}(v \oplus 1 \parallel y \parallel s) \oplus \mathrm{MAC}(v \parallel z \parallel s) \oplus \mathrm{MAC}(v \oplus 1 \parallel t \parallel s) = 0$ for any suffix s.

Therefore if we want to forge a MAC for any message m of size $\ell \geq 2$ blocks we parse it as $m = v \parallel w \parallel s$ (where s has zero, one, or several blocks) and perform

the attack to recover Δ_1 and Δ_3. Then we can forge using the previous relation, and Equation (2):

$$\begin{aligned}\mathrm{MAC}(v \parallel w \parallel s) = &\mathrm{MAC}(v \oplus 1 \parallel w \oplus \Delta_1 \parallel s) \oplus \mathrm{MAC}(v \parallel w \oplus \Delta_3 \parallel s) \\ &\oplus \mathrm{MAC}(v \oplus 1 \parallel w \oplus \Delta_1 \oplus \Delta_3 \parallel s)\end{aligned}$$

Optimizing the time complexity. Equation (2) can also be used to reduce the time complexity below 2^n, at the cost of more oracle queries. Indeed, if we consider a subset $\mathcal{C}$ of $\{0,1\}^n$, we have:

$$\begin{aligned}\mathcal{R}(x,y,z,t) &\Leftrightarrow \forall c \in \mathcal{C}, \mathcal{R}(x \oplus c, y \oplus c, z \oplus c, t \oplus c) \\ &\Rightarrow \forall c \in \mathcal{C},\ \mathrm{MAC}(\phi(x \oplus c)) \oplus \mathrm{MAC}(\psi(y \oplus c)) \\ &\qquad \oplus \mathrm{MAC}(\phi(z \oplus c)) \oplus \mathrm{MAC}(\psi(t \oplus c)) = 0 \\ &\Rightarrow \bigoplus\nolimits_{c\in\mathcal{C}} \mathrm{MAC}(\phi(x \oplus c)) \oplus \bigoplus\nolimits_{c\in\mathcal{C}} \mathrm{MAC}(\psi(y \oplus c)) \\ &\qquad \oplus \bigoplus\nolimits_{c\in\mathcal{C}} \mathrm{MAC}(\phi(z \oplus c)) \oplus \bigoplus\nolimits_{c\in\mathcal{C}} \mathrm{MAC}(\psi(t \oplus c)) = 0 \quad (4)\end{aligned}$$

If we select $\mathcal{C}$ as a linear subspace, then the last expression does not depend on the full (x,y,z,t), but only on their projection on the orthogonal of $\mathcal{C}$. Concretely, we use $\mathcal{C} = \left\{x : x_{[3n/7:n]} = 0\right\} = \left\{x : x < 2^{3n/7}\right\}$, so that the value $\bigoplus_{c\in\mathcal{C}} \mathrm{MAC}(\phi(x \oplus c))$ is independent of bits 0 to $3n/7 - 1$ of x.

Therefore, we consider the rewritten MAC function

$$\mathrm{MAC}'(v \parallel w) = \bigoplus\nolimits_{c\in\mathcal{C}} \mathrm{MAC}(v \parallel w \oplus c),$$

the following message injections, with a $4n/7$-bit input

$$\phi'(i) = 0 \parallel i|0 \qquad\qquad \psi'(i) = 1 \parallel i|0,$$

and a reduced relation over $4n/7$-bit values:

$$\mathcal{R}'(x,y,z,t) := \begin{cases} x \oplus y = (E_1(0) \oplus E_1(1))_{[3n/7:n]} \\ y \oplus z = (E_3(0) \oplus E_3(1))_{[3n/7:n]} \\ z \oplus t = (E_1(0) \oplus E_1(1))_{[3n/7:n]} \\ t \oplus x = (E_3(0) \oplus E_3(1))_{[3n/7:n]} \end{cases}$$

$$\Leftrightarrow \begin{cases} x \oplus y \oplus z \oplus t = 0 \\ x \oplus y = (E_1(0) \oplus E_1(1))_{[3n/7:n]} \\ x \oplus t = (E_3(0) \oplus E_3(1))_{[3n/7:n]} \end{cases}$$

Thanks to Equation 4, we still have:

$$\mathcal{R}'(x,y,z,t) \Rightarrow \mathrm{MAC}'(\phi'(x)) \oplus \mathrm{MAC}'(\psi'(y)) \oplus \mathrm{MAC}'(\phi'(z)) \oplus \mathrm{MAC}'(\psi'(t)) = 0$$

Since the relation $\mathcal{R}'$ is now only a $12n/7$-bit condition, we can use shorter lists than before, with just $2^{3n/7}$ elements. We can also increase the filtering using

the same trick as previously, considering the following lists:

$$L'_1 = \left\{ x \parallel \mathrm{MAC}'(\phi'(x)) \parallel \mathrm{MAC}'(\phi'(x \oplus 1)) \ : \ x \in \{0,1\}^{4n/7},\, x_{[0:n/7]} = 0 \right\}$$
$$L'_2 = \left\{ y \parallel \mathrm{MAC}'(\psi'(y)) \parallel \mathrm{MAC}'(\psi'(y \oplus 1)) \ : \ y \in \{0,1\}^{4n/7},\, y_{[n/7:2n/7]} = 0 \right\}$$
$$L'_3 = \left\{ z \parallel \mathrm{MAC}'(\phi'(z)) \parallel \mathrm{MAC}'(\phi'(z \oplus 1)) \ : \ z \in \{0,1\}^{4n/7},\, z_{[2n/7:3n/7]} = 0 \right\}$$
$$L'_4 = \left\{ t \parallel \mathrm{MAC}'(\psi'(t)) \parallel \mathrm{MAC}'(\psi'(t \oplus 1)) \ : \ t \in \{0,1\}^{4n/7},\, t_{[3n/7:4n/7]} = 0 \right\}$$

Finally, using the algorithm of Section 2.2 with $s = 3n/7$ and $p = 0$, we can locate a right quadruple using $\tilde{\mathcal{O}}(2^{6n/7})$ queries, $\tilde{\mathcal{O}}(2^{6n/7})$ operations, and $\mathcal{O}(2^{3n/7})$ memory. This recovers only $4n/7$ bits of $E_1(0) \oplus E_1(1)$ and $E_3(0) \oplus E_3(1)$, but we can easily recover the remaining bits, either by brute force, or by repeating the attack with a different set $\mathcal{C}$.

3.2 Attacking GCM-SIV2

GCM-SIV2 is an authenticated encryption mode designed by Iwata and Minematsu [20] as a double-block-hash version of GCM-SIV (in the following, we consider GCM-SIV2 with GHASH as the underlying universal hash function). For simplicity, we focus on the authentication part of GCM-SIV2, using inputs with a non-empty associated data, and an empty message. In this case, GCM-SIV2 becomes a nonce-based MAC. The message M (considered as associated data for the mode) is zero-padded, divided into n-bit blocks, and the length is appended in an extra block. Then the construction is defined as follows, with $\odot$ a finite field multiplication (see also Figure 3):

$$\begin{aligned} \Sigma(N, M) &= N \oplus \ell \odot H_1 \oplus \textstyle\bigoplus_{i=1}^{\ell} m_i \odot H_1^{\ell+2-i} \\ \Theta(N, M) &= N \oplus \ell \odot H_2 \oplus \textstyle\bigoplus_{i=1}^{\ell} m_i \odot H_2^{\ell+2-i} \\ \mathrm{MAC}(N, M) &= E_1(\Sigma(M)) \oplus E_2(\Theta(M)) \,\Big\|\, E_3(\Sigma(M)) \oplus E_4(\Theta(M)) \end{aligned}$$

Attack. The structure of the authentication part of GCM-SIV2 is essentially the same as the structure of SUM-ECBC, where the block cipher calls E_1 and E_3 are replaced by multiplication by H_1 and H_2. The finalization function has a $2n$-bit output $\mathrm{MAC}^1, \mathrm{MAC}^2$, but quadruples following $\mathcal{R}$ will collide on both outputs. Thus, we can essentially repeat the SUM-ECBC attack, but there is an important difference: GCM-SIV2 is a nonce-based MAC, rather than a deterministic one. Therefore, all queries must include a nonce N, and we should not query two different messages with the same nonce. We adapt the previous attack using message injection functions that output both a nonce and a message, so that we use two fixed messages, 0 and 1, with variable nonces:

$$\phi(i) = (i, 0) \qquad\qquad \psi(i) = (i, 1)$$

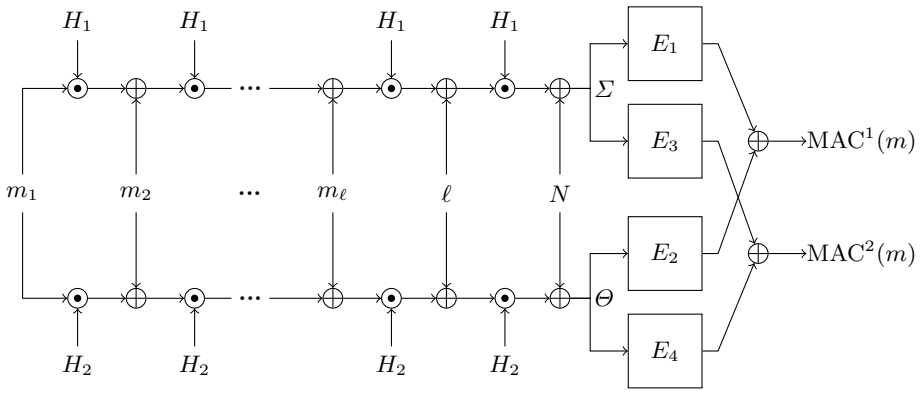


Fig. 3. Diagram for authentication in `GCM-SIV2` using `GHASH` with a ℓ-block message, a nonce N, hash keys H_1 and H_2.

$$\mathrm{MAC}(\phi(i)) = E_1\big(\underbrace{i \oplus H_1}_{\Sigma_0(i)}\big) \oplus E_2\big(\underbrace{i \oplus H_2}_{\Theta_0(i)}\big) \;\Big\|\; E_3\big(\Sigma_0(i)\big) \oplus E_4\big(\Theta_0(i)\big)$$

$$\mathrm{MAC}(\psi(i)) = E_1\big(\underbrace{i \oplus H_1 \oplus H_1^2}_{\Sigma_1(i)}\big) \oplus E_2\big(\underbrace{i \oplus H_2 \oplus H_2^2}_{\Theta_1(i)}\big) \;\Big\|\; E_3\big(\Sigma_1(i)\big) \oplus E_4\big(\Theta_1(i)\big).$$

We consider quadruples of nonce/messages X, Y, Z, T with

$$X = \phi(x) \qquad Y = \psi(y) \qquad Z = \phi(z) \qquad T = \psi(t),$$

and we have the same kind of relations as in the previous attack:

$$\mathcal{R}(x,y,z,t) := \begin{cases} \Sigma_0(x) = \Sigma_1(y) \\ \Sigma_0(z) = \Sigma_1(t) \\ \Theta_0(z) = \Theta_1(y) \\ \Theta_0(x) = \Theta_1(t). \end{cases} \Leftrightarrow \begin{cases} x \oplus y \oplus z \oplus t = 0 \\ x \oplus y = H_1^2 \\ x \oplus t = H_2^2 \end{cases}$$

$$\Rightarrow \mathrm{MAC}(\phi(x)) \oplus \mathrm{MAC}(\psi(y)) \oplus \mathrm{MAC}(\phi(z)) \oplus \mathrm{MAC}(\psi(t)) = 0$$

Since the MAC output is $2n$-bit long, we can directly build an attack with $\mathcal{O}(2^{3n/4})$ queries: we consider four distinct sets $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{T}$ of $2^{3n/4}$ values, and we look for a quadruple $(x, y, z, t) \in \mathcal{X} \times \mathcal{Y} \times \mathcal{Z} \times \mathcal{T}$, such that

$$\begin{cases} x \oplus y \oplus z \oplus t = 0 \\ \mathrm{MAC}(\phi(x)) \oplus \mathrm{MAC}(\psi(y)) \oplus \mathrm{MAC}(\phi(z)) \oplus \mathrm{MAC}(\psi(t)) = 0 \end{cases} \tag{5}$$

we expect to find one good quadruple that respects $\mathcal{R}$ along with $\mathcal{O}(1)$ quadruples that randomly satisfy the observable filter (5). This leads to an attack with $\mathcal{O}(2^{3n/4})$ queries and time $\tilde{O}(2^{3n/2})$. Since we recover H_1 and H_2 (from $H_1^2 = x \oplus y$ and $H_2^2 = x \oplus t$), we can do universal forgeries. In addition, we can also easily adapt the attack with $\mathcal{O}(2^{6n/7})$ queries and time $\tilde{O}(2^{6n/7})$.

4 Attacking PMAC-like constructions

We now describe attacks against PMAC+ [43] and related constructions: 1kMAC+ [9], and LightMAC+ [33]. We have an existential forgery attack with $\mathcal{O}(2^{3n/4})$ queries and $\tilde{\mathcal{O}}(2^{3n/2})$ operations (using memory $\mathcal{O}(2^{3n/4})$), with a range of time-memory trade-offs with $\mathcal{O}(2^t)$ queries, with $3n/4 < t < n$, and $\tilde{\mathcal{O}}(2^{3n-2t})$ operations (using memory $\mathcal{O}(2^t)$).

4.1 Attacking PMAC+

PMAC+ was designed by Yasuda in 2011 [43], as a variant of PMAC [5] with a larger internal state. The scheme internally uses a tweakable block cipher construction inspired by the XE construction [39], that we denote as $\tilde{E}_i$. The message M is first padded with 10^* padding, and divided into n-bit blocks, but for simplicity we ignore the padding in our description. The construction is shown in Figure 4[3]:

$$\begin{aligned} \Sigma(M) &= \bigoplus_{i=1}^{\ell} \tilde{E}_i(m_i) & \tilde{E}_i(x) &= E_1(x \oplus 2^i \odot \Delta_0 \oplus 2^{2i} \odot \Delta_1) \\ \Theta(M) &= \bigoplus_{i=1}^{\ell} 2^{\ell-i} \odot \tilde{E}_i(m_i) & \Delta_0 &= E_1(0) \quad \Delta_1 = E_1(1) \\ \mathrm{MAC}(M) &= E_2(\Sigma(M)) \oplus E_3(\Theta(M)) \end{aligned}$$

Attack. As in the previous attack, we use message injection functions with two different prefixes, but we include an extra block u to define related quadruples:

$$\phi_u(i) = u \,\|\, 0 \,\|\, i \qquad\qquad \psi_u(i) = u \,\|\, 1 \,\|\, i$$

$$\mathrm{MAC}(\phi_u(i)) = E_2\Big(\underbrace{\tilde{E}_1(u) \oplus \tilde{E}_2(0) \oplus \tilde{E}_3(i)}_{\Sigma_{u,0}(i)}\Big) \oplus E_3\Big(\underbrace{4\tilde{E}_1(u) \oplus 2\tilde{E}_2(0) \oplus \tilde{E}_3(i)}_{\Theta_{u,0}(i)}\Big)$$

$$\mathrm{MAC}(\psi_u(i)) = E_2\Big(\underbrace{\tilde{E}_1(u) \oplus \tilde{E}_2(1) \oplus \tilde{E}_3(i)}_{\Sigma_{u,1}(i)}\Big) \oplus E_3\Big(\underbrace{4\tilde{E}_1(u) \oplus 2\tilde{E}_2(1) \oplus \tilde{E}_3(i)}_{\Theta_{u,1}(i)}\Big).$$

Next, we build quadruples of messages X, Y, Z, T with

$$X = \phi_u(x) \qquad Y = \psi_u(y) \qquad Z = \phi_u(z) \qquad T = \psi_u(t),$$

and we look for a quadruple with partial state collisions for the underlying pairs, *i. e.* a quadruple following the relation:

$$\mathcal{R}(x, y, z, t) := \begin{cases} \Sigma_{u,0}(x) = \Sigma_{u,1}(y) \\ \Sigma_{u,0}(z) = \Sigma_{u,1}(t) \\ \Theta_{u,0}(z) = \Theta_{u,1}(y) \\ \Theta_{u,0}(x) = \Theta_{u,1}(t). \end{cases}$$

[3] The algorithm and the figure given in [43] differ in the coefficients used to compute Θ. We use the algorithmic description because it matches later PMAC+ variants, but the attack can easily be adapted to the other case.

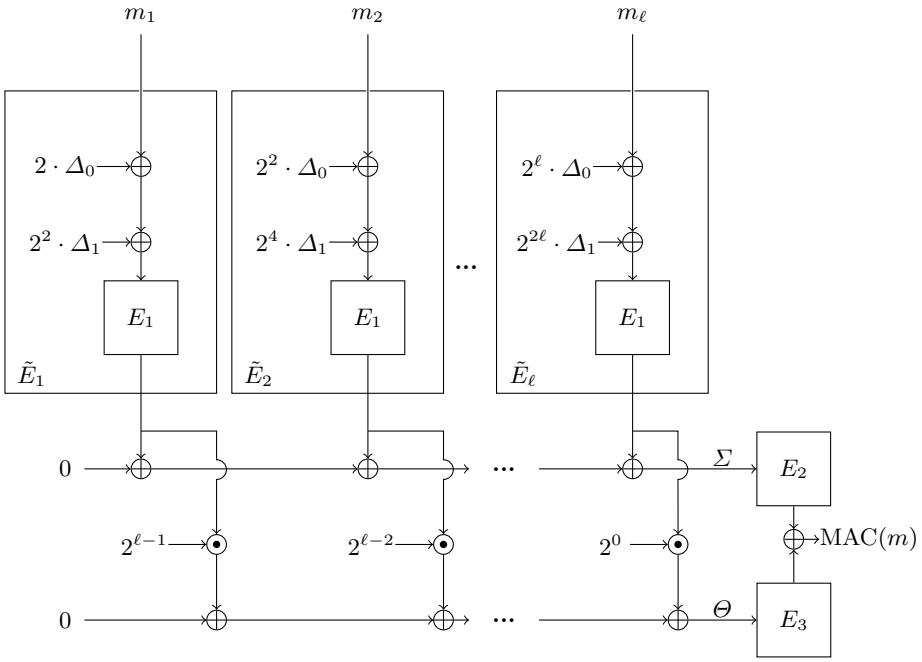


Fig. 4. Diagram for PMAC+ with a ℓ-block message where $\Delta_0 = E_1(0)$ and $\Delta_1 = E_1(1)$.

We have

$$\mathcal{R}(x,y,z,t) \Leftrightarrow \begin{cases} \tilde{E}_3(x) \oplus \tilde{E}_2(0) = \tilde{E}_3(y) \oplus \tilde{E}_2(1) \\ \tilde{E}_3(z) \oplus \tilde{E}_2(0) = \tilde{E}_3(t) \oplus \tilde{E}_2(1) \\ \tilde{E}_3(y) \oplus 2\tilde{E}_2(1) = \tilde{E}_3(z) \oplus 2\tilde{E}_2(0) \\ \tilde{E}_3(t) \oplus 2\tilde{E}_2(1) = \tilde{E}_3(x) \oplus 2\tilde{E}_2(0) \end{cases}$$

$$\Leftrightarrow \begin{cases} \tilde{E}_3(x) \oplus \tilde{E}_3(y) \oplus \tilde{E}_3(z) \oplus \tilde{E}_3(t) = 0 \\ \tilde{E}_3(x) \oplus \tilde{E}_3(y) = \tilde{E}_2(0) \oplus \tilde{E}_2(1) \\ \tilde{E}_3(t) \oplus \tilde{E}_3(x) = 2\tilde{E}_2(0) \oplus 2\tilde{E}_2(1) \end{cases}$$

Again, $\mathcal{R}$ defines a $3n-$bit relation, and we can detect it through the sum of the MACs following Equation (1):

$$\mathcal{R}(x,y,z,t) \Rightarrow \mathrm{MAC}(\phi_u(x)) \oplus \mathrm{MAC}(\psi_u(y)) \oplus \mathrm{MAC}(\phi_u(z)) \oplus \mathrm{MAC}(\psi_u(t)) = 0$$

In addition, the relation $\mathcal{R}$ is independent of the value u, so that we can easily build several quadruples that satisfy $\mathcal{R}$ simultaneously. This leads to an attack with $\mathcal{O}(2^{3n/4})$ queries: we consider four sets $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{T}$ of $2^{3n/4}$ random values, and we look for a quadruple $(x,y,z,t) \in \mathcal{X} \times \mathcal{Y} \times \mathcal{Z} \times \mathcal{T}$, such that

$$\forall u \in \{0,1,2\},\ \mathrm{MAC}(\phi_u(x)) \oplus \mathrm{MAC}(\psi_u(y)) \oplus \mathrm{MAC}(\phi_u(z)) \oplus \mathrm{MAC}(\psi_u(t)) = 0$$

We expect on average one random quadruple (with 2^{3n} potential quadruples, and a $3n$-bit filtering), and one quadruple satisfying $\mathcal{R}$ (also a $3n$-bit condition). The correct quadruple can easily be checked with a few extra queries.

In practice, we use the generalized birthday algorithms of Section 2.2 in order to optimize the complexity of the attack. We consider four lists:

$$\begin{aligned}
L_1 &= \{\mathrm{MAC}(\phi_0(x)) \,\|\, \mathrm{MAC}(\phi_1(x)) \,\|\, \mathrm{MAC}(\phi_2(x)) \;:\; x \in \mathcal{X}\} \\
L_2 &= \{\mathrm{MAC}(\psi_0(y)) \,\|\, \mathrm{MAC}(\psi_1(y)) \,\|\, \mathrm{MAC}(\psi_2(y)) \;:\; y \in \mathcal{Y}\} \\
L_3 &= \{\mathrm{MAC}(\phi_0(z)) \,\|\, \mathrm{MAC}(\phi_1(z)) \,\|\, \mathrm{MAC}(\phi_2(z)) \;:\; z \in \mathcal{Z}\} \\
L_4 &= \{\mathrm{MAC}(\psi_0(t)) \,\|\, \mathrm{MAC}(\psi_1(t)) \,\|\, \mathrm{MAC}(\psi_2(t)) \;:\; t \in \mathcal{T}\}
\end{aligned}$$

and we look for a quadruple $(x, y, z, t) \in \mathcal{X} \times \mathcal{Y} \times \mathcal{Z} \times \mathcal{T}$ such that $L_1[x] \oplus L_2[y] \oplus L_3[z] \oplus L_4[t] = 0$. This can be done with $\tilde{\mathcal{O}}(2^{3n/2})$ operations, using a memory of size $\mathcal{O}(2^{3n/4})$. Finally, once a quadruple (x, y, z, t) satisfying $\mathcal{R}(x, y, z, t)$ has been detected, it can be used to generate forgeries. Indeed, we can predict the MAC of a new message by making three new queries using Equation (1):

$$\forall u,\ \mathrm{MAC}(\phi_u(x)) = \mathrm{MAC}(\psi_u(y)) \oplus \mathrm{MAC}(\psi_u(z)) \oplus \mathrm{MAC}(\phi_u(t))$$

Time-Query Trade-offs. As opposed to the `SUM-ECBC` attack, we don't have an analogue to Equation (2) that can be used to reduce the time complexity. However, the time complexity of the algorithm can be slightly reduced when using more than $\mathcal{O}(2^{3n/4})$ queries. If we consider sets $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{T}$ of size 2^t with $3n/4 < t < n$, the resulting 4-sum is slightly easier, because there are 2^{4t-3n} expected solutions. Using the algorithm of section 2.2, this can be solved in time $\tilde{\mathcal{O}}(2^{3n-2t})$, using a memory of size $\mathcal{O}(2^t)$.

4.2 Attacking `LightMAC+`

`LightMAC+` was designed by Naito [33] using ideas from `PMAC+` [43] and `LightMAC` [29]. If we consider it as based on a tweakable block cipher $\tilde{E}$, it follows the same structure as `PMAC+` (see Figure 5), but $\tilde{E}$ takes a message block smaller than n bits:

$$\begin{aligned}
\Sigma(M) &= \textstyle\bigoplus_{i=1}^{\ell} \tilde{E}_i(m_i) & \tilde{E}_i(x) &= E_1(i|x) \\
\Theta(M) &= \textstyle\bigoplus_{i=1}^{\ell} 2^{\ell-i} \odot \tilde{E}_i(m_i) & & \\
\mathrm{MAC}(M) &= E_2(\Sigma(M)) \oplus E_3(\Theta(M)) & &
\end{aligned}$$

Since the structure of `LightMAC+` is the same as the structure of `PMAC+`, we can use the same attack. The only difference from our point of view is that the message blocks are shorter than the block-size. As long as one message block is big enough to fit $2^{3n/4}$ different values, our attack will succeed.

This attack violates the improved security proof recently published at CT-RSA [34], with a security bound of $\mathcal{O}(q_t^2 q_v / 2^{2n})$ (with q_t MAC queries and q_v verification queries). Indeed, our attack reaches a constant success probability with $q_t = \mathcal{O}(2^{3n/4})$ and $q_v = 1$. We have shared our attack with Naito and he agreed that his proof is flawed.

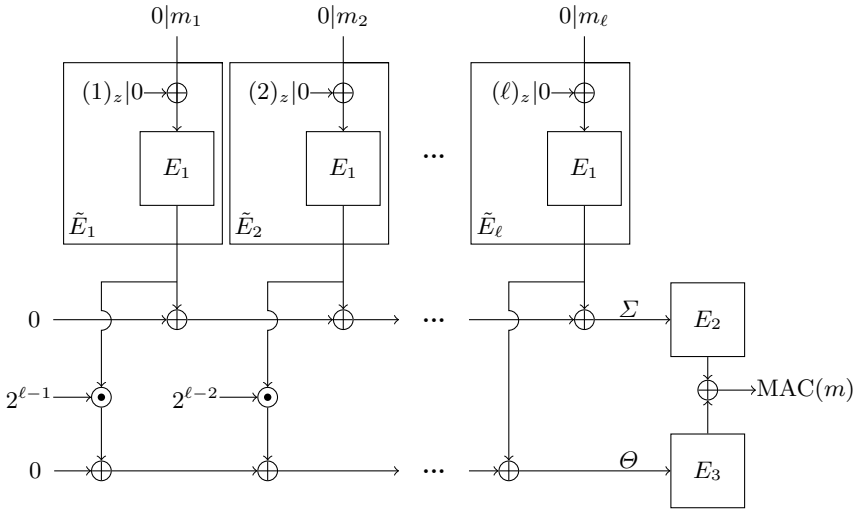


Fig. 5. Diagram for `LightMAC+` with $(n-z)$-bit blocks of a ℓ-block message where $(v)_z$ is the value v written over z bits.

4.3 Attacking `1kPMAC+`

`1kPMAC+` is a single-key variant of `PMAC+` [43] designed by Datta, Dutta, Nandi, Paul and Zhang [9], shown in Figure 8.

Since the structure of `1kPMAC+` is the same as the structure of `PMAC+`, we can use the same attack. Alternatively, we can take advantage of the `fix` functions to mount a more straightforward attack, as shown in Section 6.

5 Attacking `f9`-like constructions

Our third attack is applicable to `3kf9` [44] and similar constructions. We have a universal forgery attack with $\mathcal{O}(2^{3n/4})$ queries and $\tilde{\mathcal{O}}(2^{5n/4})$ operations using memory $\mathcal{O}(2^n)$, with a possible time-memory trade-offs.

5.1 Attacking `3kf9`

`3kf9` [44], designed by Xhang, Wu, Sui and Wang, is a three-key variant of the `f9` mode used in 3G telephony. While the original `f9` does not have security beyond the birthday bound [24], `3kf9` is secure up to $2^{2n/3}$ queries. We describe `3kf9` in Figure 6:

$$\Sigma(M) = \sigma_\ell \qquad \sigma_0 = 0 \qquad \sigma_i = E_1(\sigma_{i-1} \oplus m_i)$$
$$\Theta(M) = \bigoplus_{i=1}^{\ell} \sigma_i$$
$$\mathrm{MAC}(M) = E_2(\Sigma(M)) \oplus E_3(\Theta(M))$$

Attack. Our attack follows the same structure as the previous attacks. We start with messages of the form:

$$\phi(i) = 0 \,\|\, i \qquad\qquad \psi(i) = 1 \,\|\, i,$$

and the corresponding MACs:

$$\mathrm{MAC}(\phi(i)) = E_2\Big(\underbrace{E_1\big(x \oplus E_1(0)\big)}_{\Sigma_0(x)}\Big) \oplus E_3\Big(\underbrace{E_1\big(x \oplus E_1(0)\big) \oplus E_1\big(0\big)}_{\Theta_0(x)}\Big)$$
$$\mathrm{MAC}(\psi(i)) = E_2\Big(\underbrace{E_1\big(x \oplus E_1(1)\big)}_{\Sigma_1(x)}\Big) \oplus E_3\Big(\underbrace{E_1\big(x \oplus E_1(1)\big) \oplus E_1\big(1\big)}_{\Theta_1(x)}\Big).$$

We use quadruples of messages X, Y, Z, T with

$$X = \phi(x) \qquad Y = \psi(y) \qquad Z = \phi(z) \qquad T = \psi(t),$$

and we look for a quadruple with partial state collisions for the underlying pairs, *i. e.* a quadruple following the relation:

$$\begin{aligned}
\mathcal{R}(x,y,z,t) := &\begin{cases} \Sigma_0(x) = \Sigma_1(y) \\ \Sigma_0(z) = \Sigma_1(t) \\ \Theta_0(z) = \Theta_1(y) \\ \Theta_0(x) = \Theta_1(t). \end{cases} \\
\Leftrightarrow &\begin{cases} x \oplus E_1(0) = y \oplus E_1(1) \\ z \oplus E_1(0) = t \oplus E_1(1) \\ E_1(z \oplus E_1(0)) \oplus E_1(0) = E_1(y \oplus E_1(1)) \oplus E_1(1) \\ E_1(x \oplus E_1(0)) \oplus E_1(0) = E_1(t \oplus E_1(1)) \oplus E_1(1) \end{cases} \\
\Leftrightarrow &\begin{cases} x \oplus y \oplus z \oplus t = 0 \\ x \oplus y = E_1(0) \oplus E_1(1) \\ E_1(x \oplus E_1(0)) \oplus E_1(t \oplus E_1(1)) = E_1(0) \oplus E_1(1) \end{cases} \\
\Rightarrow &\ \mathrm{MAC}(\phi(x)) \oplus \mathrm{MAC}(\psi(y)) \oplus \mathrm{MAC}(\phi(z)) \oplus \mathrm{MAC}(\psi(t)) = 0.
\end{aligned}$$

As in the previous attacks, $\mathcal{R}$ defines a $3n-$bit relation. Moreover, we can easily observe when $x \oplus y \oplus z \oplus t = 0$, and the relation $x \oplus y = E_1(0) \oplus E_1(1)$ can be verified across several quadruples. We don't have related quadruples satisfying $\mathcal{R}$ simultaneously as in the previous attacks, but we can use those properties to detect right quadruples. This leads to an attack with $\tilde{\mathcal{O}}(2^{3n/4})$ queries: we consider four sets $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{T}$ of $\sqrt[4]{n} \times 2^{3n/4}$ random values, and we look for quadruples $(x, y, z, t) \in \mathcal{X} \times \mathcal{Y} \times \mathcal{Z} \times \mathcal{T}$, such that:

$$\begin{cases} x \oplus y \oplus z \oplus t = 0 \\ \mathrm{MAC}(\phi(x)) \oplus \mathrm{MAC}(\psi(y)) \oplus \mathrm{MAC}(\phi(z)) \oplus \mathrm{MAC}(\psi(t)) = 0. \end{cases} \tag{6}$$

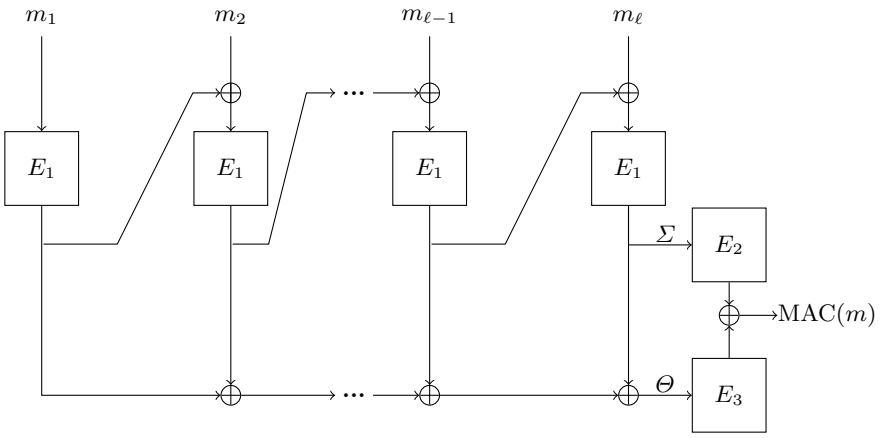


Fig. 6. Diagram for `3kf9` with a $\ell-$block message.

Since this a $2n$-bit condition, we expect on average $n \cdot 2^n$ quadruples (x, y, z, t) satisfying (6). In order to filter out the right ones, we look at the value $x \oplus y$ for all these quadruples. While the wrong quadruples should have a random $x \oplus y$, the right ones have $x \oplus y = E_1(0) \oplus E_1(1)$. Therefore, with high probability, the most frequent value for $x \oplus y$ is equal to $E_1(0) \oplus E_1(1)$, and quadruples satisfying this extra relation are right quadruples with probability $1/2$. More precisely, we expect on average n wrong quadruples for each value of $x \oplus y$, and n right quadruples with $x \oplus y = E_1(0) \oplus E_1(1)$.

Optimizing the time complexity. While the algorithm of Section 2.2 would take time $\tilde{\mathcal{O}}(2^{3n/2})$ with $\tilde{\mathcal{O}}(2^{3n/4})$ queries, we can reduce the time complexity using sets $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{T}$ with some structure. More precisely, we use:

$$\mathcal{X} = \mathcal{Z} = \left\{x \in \{0,1\}^n \ : \ x_{[0:n/4]} = 0\right\}$$
$$\mathcal{Y} = \mathcal{T} = \left\{x \in \{0,1\}^n \ : \ x_{[n/4:n/2]} = 0\right\}$$

so that quadruples can be written as

$$x =: x_3|x_2|x_1|0 \in \mathcal{X} \qquad y =: y_3|y_2|0|y_0 \in \mathcal{Y}$$
$$z =: z_3|z_2|z_1|0 \in \mathcal{Z} \qquad t =: t_3|t_2|0|t_0 \in \mathcal{T}.$$

In particular, right quadruples satisfy $x \oplus y \oplus z \oplus t = 0$, therefore $x_1 = z_1$, $y_0 = t_0$, and $x_3|x_2 \oplus z_3|z_2 = y_3|y_2 \oplus t_3|t_2$. We use these properties to adapt the algorithm of Section 2.2 and locate the quadruples efficiently. First we guess the $n/2$-bit value $\alpha_3|\alpha_2 := x_3|x_2 \oplus z_3|z_2 = y_3|y_2 \oplus t_3|t_3$. Then, for each $x = x_3|x_2|x_1|0$, there is a single candidate $z = (x_3 \oplus \alpha_3)|(x_2 \oplus \alpha_2)|x_1|0$ that could be part of a right quadruple. Similarly, every $y = y_3|y_2|0|y_0$ can be paired with a single

$t = (y_3 \oplus \alpha_3)|(y_2 \oplus \alpha_2)|0|y_0$. Therefore, we consider the two following lists:

$$L_1 = \{\mathrm{MAC}(\phi(x_3|x_2|x_1|0)) \oplus \mathrm{MAC}((x_3 \oplus \alpha_3)|(x_2 \oplus \alpha_2)|x_1|0) \ : \ x_3|x_2|x_1|0 \in \mathcal{X}\}$$
$$L_2 = \{\mathrm{MAC}(\phi(y_3|y_2|0|y_0)) \oplus \mathrm{MAC}((y_3 \oplus \alpha_3)|(y_2 \oplus \alpha_2)|0|y_0) \ : \ y_3|y_2|0|y_0 \in \mathcal{Y}\}$$

After sorting the lists, we look for matches, and the corresponding quadruples x, y, z, t are exactly the quadruples satisfying

$$\begin{cases} x \oplus y \oplus z \oplus t = 0 \\ (x \oplus z)_{[n/2:n]} = \alpha_3|\alpha_2 \\ \mathrm{MAC}(\phi(x)) \oplus \mathrm{MAC}(\psi(y)) \oplus \mathrm{MAC}(\phi(z)) \oplus \mathrm{MAC}(\psi(t)) = 0. \end{cases} \tag{7}$$

More precisely, a match $L_1[x] = L_2[y]$ suggests $z = x \oplus \alpha_3|\alpha_2|0|0$ and $t = y \oplus \alpha_3|\alpha_2|0|0$, but there are four corresponding quadruples: (x, y, z, t), (z, y, x, t), (x, t, z, y), (z, t, x, y), and two candidate values for $E_1(0) \oplus E_1(1)$: $x \oplus y$ and $x \oplus y \oplus \alpha_3|\alpha_2|0|0$.

We need $\tilde{\mathcal{O}}(2^{3n/4})$ operations to generate those quadruples. We repeat this $2^{n/2}$ times to exhaust all $n/2$-bit values $\alpha_3|\alpha_2$ and generate all quadruples satisfying (6). Finally, we use an array to count the number of occurrences of each possible value of $x \oplus y$. Each counter receives an average two values, but the counter corresponding to $E_1(0) \oplus E_1(1)$ will receive three values on average. After repeating all the operations $\mathcal{O}(n)$ times, with some arbitrary constants in place of the zero bits, the highest counter corresponds to $E_1(0) \oplus E_1(1)$ with high probability, as proved in Section 5.2. This gives an attack with $\tilde{\mathcal{O}}(2^{3n/4})$ queries, $\tilde{\mathcal{O}}(2^{5n/4})$ operations, and $\mathcal{O}(2^n)$ memory[4].

Time-Memory Trade-offs. We can reduce the memory usage if we store only a subset of the counters, and repeat the whole algorithm until the whole set has been covered. Concretely, we store only the counters with a fixed value for bits $[0 : n/8]$ and $[n/4 : 3n/8]$ of $x \oplus y$. Because of the way the lists L_1 and L_2 are constructed, we have actually fixed $n/8$ bits of y_0 and x_1, and we can reduce the lists to size $2^{5n/8}$. Therefore we evaluate $2^{3n/4}$ counters in time $\tilde{\mathcal{O}}(2^{n/2} \cdot 2^{5n/8})$, using only $\mathcal{O}(2^{3n/4})$ memory. We repeat iteratively over the full counter set, so we need time $\tilde{\mathcal{O}}(2^{n/4} \cdot 2^{n/2} \cdot 2^{5n/8}) = \tilde{\mathcal{O}}(2^{11n/8})$. More generally, we have a time-memory trade-off with time $\tilde{\mathcal{O}}(2^{5n/4+t/2})$ and memory $\mathcal{O}(2^{n-t})$ for $0 < t < n/4$.

Forgeries. Once we found a quadruple (x, y, z, t) that respects $\mathcal{R}(x, y, z, t)$ we know that after processing message $\phi(x) = 0 \parallel x$ and $\psi(t) = 1 \parallel t$, there is no difference in the Θ part of the state ($\Theta_0(x) = \Theta_1(t)$). Moreover we have $\Theta_0(x) = \Sigma_0(x) \oplus E_1(0)$ and $\Theta_1(t) = \Sigma_1(x) \oplus E_1(1)$; this implies that there is a difference $E_1(0) \oplus E_1(1) = x \oplus y$ in the Σ part of the state. Therefore, we can build a full state collision with message $0 \parallel x \parallel 0$ and $1 \parallel t \parallel x \oplus y$. In particular, the following relation can be used to create forgeries with an arbitrary message m (of any length):

$$\mathrm{MAC}(0 \parallel x \parallel 0 \parallel m) = \mathrm{MAC}(1 \parallel t \parallel x \oplus y \parallel m).$$

[4] We can actually reduce the polynomial factors by fixing only $(n - \log_2(n))/4$ bits to zero, in order to have sets of size $\sqrt[4]{n} \cdot 2^{3n/4}$.

Universal Forgeries. We can even forge the tag of an arbitrary message of length at least $(2n+2)$ blocks with complexity only $n+1$ times the complexity of the simple forgery attack. The technique is more advanced and inspired by the multi-collision attack described by Joux [23]. For ease of notation we'll show how to forge the signature for a message starting with $2n+2$ blocks of zero, but this can be trivially adapted for any message.

First, we find a quadruple (x_1, y_1, z_1, t_1) as before. Then we consider messages $0\|0$ and $1\|x_1 \oplus y_1$. Since $x_1 \oplus y_1 = E_1(0) \oplus E_1(1)$, we have $\Sigma(0\|0) = \Sigma(1\|x_1 \oplus y_1)$, *i. e.* the Σ part of the state collides. Moreover, we know the difference in the Θ part: $\Theta(0 \parallel 0) \oplus \Theta(1 \parallel x_1 \oplus y_1) = x_1 \oplus y_1$.

More generally, at step i we use message injection functions

$$\phi_i(x) = \underbrace{0 \parallel 0 \parallel \ldots \parallel 0}_{\times 2(i-1)} \parallel 0 \parallel x \qquad \psi_i(x) = \underbrace{0 \parallel 0 \parallel \ldots \parallel 0}_{\times 2(i-1)} \parallel 1 \parallel x,$$

to look for a quadruple of messages

$$X_i = \phi_i(x_i) \qquad Y_i = \psi_i(y_i) \qquad Z_i = \phi_i(z_i) \qquad T_i = \psi_i(t_i).$$

When a right quadruple (x_i, y_i, z_i, t_i) has been identified, we can deduce that the MACs for $0 \parallel 0 \parallel \ldots \parallel 0 \parallel 0 \parallel 0$ and $0 \parallel 0 \parallel \ldots \parallel 0 \parallel 1 \parallel x_i \oplus y_i$ will match on the Σ branch and differ by $x_i \oplus y_i$ in their Θ branch.

After several iterations, we have actually built a multi-collision: all the messages $h_1 \parallel h_2 \parallel \ldots \parallel h_n \parallel h_{n+1}$ with $h_i \in \{(1 \parallel x_i \oplus y_i), (0 \parallel 0)\}$ collide on the Σ branch. In addition, we also know the difference in the Θ branch for those messages: it is equal to $\bigoplus_{\{i\,:\,h_i \neq 0\|0\}} (x_i \oplus y_i)$.

After at most $n+1$ steps, we can find a non empty subset $\mathcal{I} \subseteq [1 : n+1]$ such that $\bigoplus_{i \in \mathcal{I}} (x_i \oplus y_i) = 0$ by simple linear algebra[5]. This gives a collision on the full state, using messages $m_0 = 0 \parallel 0 \parallel \ldots \parallel 0$ (with $2(n+1)$ blocks) and $h = h_1 \parallel h_2 \parallel \ldots \parallel h_n \parallel h_{n+1}$ with $h_i = 1 \parallel x_i \oplus y_i$ if $i \in \mathcal{I}$, $h_i = 0 \parallel 0$ otherwise. Since the full state collides, we have for any message m (of any length):

$$\mathrm{MAC}(h \parallel m) = \mathrm{MAC}(m_0 \parallel m).$$

5.2 Detailed Complexity Analysis

We want to prove the claim that one will need to find $\mathcal{O}(n \cdot 2^n)$ quadruples in order to finish the attack on `3kf9` described in Section 5.1. We say the attack finishes when we recover the target value $T = E(0) \oplus E(1)$.

Assuming that each quadruple we find respects $\mathcal{R}$ with probability $1/2^n$, we fill a list of counters for every suspected values of T; a random quadruple gives two random values and a right one gives one value equal to T and one random value. Therefore we sum up the distribution of an observable value x as:

$$x \begin{cases} \xleftarrow{\$} \{0,1\}^n & \text{with probability } 1 - 1/2^{n+1} \\ \longleftarrow T & \text{with probability } 1/2^{n+1} \end{cases}$$

[5] We construct the kernel of the linear function $\lambda_i \mapsto \bigoplus_i \lambda_i (x_i \oplus y_i)$

Let N be the number of observed values, and X_i^c represents the indicator that the i^{th} value equals c (following a Bernoulli distribution), so that the counter corresponding to c is $X^c = \sum_{i=1}^{N} X_i^c$. Now we have to discriminate between the distributions of X^c with $c \neq T$, and the distribution of X^T:

$$\begin{aligned}
\Pr(X_i^T = 1) = \Pr(x = T) &= (1 - 1/2^{n+1})/2^n + 1/2^{n+1} = (3/2 - 1/2^{n+1})/2^n \\
&\implies \mathbf{E}[X^T] = N(3/2 - 1/2^{n+1})/2^n \\
\Pr(X_i^c = 1) = \Pr(x = c) &= (1 - 1/2^{n+1})/2^n \\
&\implies \mathbf{E}[X^c] = N(1 - 1/2^{n+1})/2^n \\
&\implies \mathbf{E}[X^T] \geq 3/2 \cdot \mathbf{E}[X^c]
\end{aligned}$$

We use the Chernoff bound to get a lower bound on the probability that a given counter is higher than the average value of X^T:

$$\Pr(X^c \geq \mathbf{E}[X^T]) \leq Pr(X^c \geq 3/2 \cdot \mathbf{E}[X^c]) \leq e^{-N(1-1/2^{n+1})/2^{n+1}}$$

and assuming the counters are independent:

$$\begin{aligned}
Pr(X^c < \mathbf{E}[X^T]) &\geq 1 - e^{-N(1-1/2^{n+1})/2^{n+1}} \\
Pr(\forall c \neq T : X^c < \mathbf{E}[X^T]) &\geq (1 - e^{-N(1-1/2^{n+1})/2^{n+1}})^{2^n}
\end{aligned}$$

This expression will asymptotically converge to a strictly positive constant when $e^{-N(1-1/2^{n+1})/2^{n+1}} \simeq 2^{-n}$. Therefore, we use

$$N \simeq n \ln(2) \cdot \frac{2^{n+1}}{(1 - 1/2^{n+1})} = \mathcal{O}(n \cdot 2^n).$$

Since we observe 2 values per quadruples, this makes $\mathcal{O}(n \cdot 2^n)$ quadruples. Moreover, the event '$X^T \geq \mathbf{E}[X^T]$' has a probability close to 0.5, therefore after $\mathcal{O}(n \cdot 2^n)$ quadruples, we indeed have a $\Omega(1)$ probability that X^T is greater than all of the other counters, which allows to recover the value T. Performing the attack until the end with probability $\Omega(1)$ also requires $\mathcal{O}(n \cdot 2^n)$ quadruples.

To get to this result some assumptions have been made, like the independence of the counters, but they all tend to be either conservative or asymptotically true.

5.3 Attacking `1kf9`

`1kf9` is a single-key variant of `3kf9` suggested in [8], and later withdrawn. Since the structure of `1kf9` is the same as the structure of `3kf9`, we can use the same attack. However, in the next section, we give an attack with birthday complexity using properties of the `fix` functions.

6 Attacks using collision in `fix` functions

Finally, we show attacks against single key variant of beyond-birthday-bound MACs based on `fix` functions, as defined by by Datta, Dutta, Nandi, Paul and

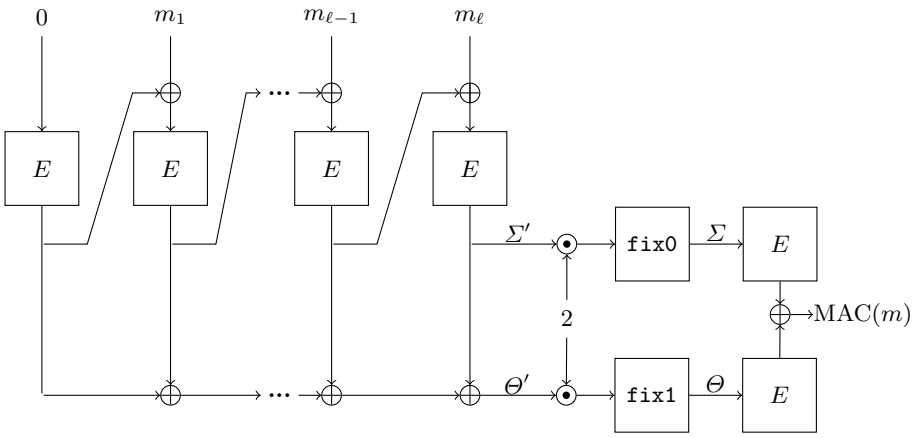


Fig. 7. Diagram for `1kf9` with a $\ell-$block message.

Zhang [8,9]. The `fix` functions just fix the least significant bit an n-bit value to zero or one, and are used for domain separation:

$$\texttt{fix0} : x \mapsto x_{[1:n]}|0 \qquad\qquad \texttt{fix1} : x \mapsto x_{[1:n]}|1$$

Datta *et al.* used those function to build a single-key variant of `PMAC+` called `1kPMAC+` [9], and a single-key variant of `3kf9` called `1kf9` [8], both with security up to $2^{2n/3}$ queries. However, `1kf9` has been withdrawn because of issues in its security proof. In this section, we exploit trivial collisions in the `fix` functions to build colliding pairs or quadruples more easily:

$$\texttt{fix0}(x) = \texttt{fix0}(x \oplus 1) \qquad\qquad \texttt{fix1}(x) = \texttt{fix1}(x \oplus 1)$$

This allows a more straightforward attack against `1kPMAC+` with the same complexity as the attacks in Section 4, and an attack against `1kf9` [8] with birthday complexity, violating its security claims.

6.1 Attacking `1kf9`

The `1kf9` mode uses the `fix` function for domain separation to build a single-key variant of `3kf9`, as shown in Figure 7:

$$\begin{aligned} \sigma_0 &= 0 & \sigma_i &= E(\sigma_{i-1} \oplus m_i) \\ \Sigma'(M) &= \sigma_\ell & \Sigma(M) &= 2 \odot \texttt{fix0}(\Sigma'(M)) \\ \Theta'(M) &= \textstyle\bigoplus_{i=1}^{\ell} \sigma_i & \Theta(M) &= 2 \odot \texttt{fix1}(\Theta'(M)) \\ \mathrm{MAC}(M) &= E(\Sigma(M)) \oplus E(\Theta(M)) \end{aligned}$$

Attack. Because of a mistake in the proof of `1kf9`, we can use pairs of messages instead of quadruples. More precisely, instead of looking for a quadruple with pairwise collisions in Σ and Θ, we look for a pair of message X, Y colliding on Σ', and with a difference in Θ' that will be absorbed by the `fix1` function. Therefore, we define the relation $\mathcal{R}$ as:

$$\mathcal{R}(X,Y) := \begin{cases} \Sigma'(X) = \Sigma'(Y) \\ 2\Theta'(X) = 2\Theta'(Y) \oplus 1 \end{cases} \\ \Rightarrow \mathrm{MAC}(X) = \mathrm{MAC}(Y).$$

We build the messages with different postfixes, parametrized by u:

$$X = \phi_u(x) = x \parallel u \qquad\qquad Y = \psi_u(y) = y \parallel u \oplus d,$$

where d is the inverse of 2 in the finite field. With this construction, we have

$$\begin{aligned} \Sigma'(\phi_u(x)) &= E\big(u \oplus E(x \oplus E(0))\big) \\ \Theta'(\phi_u(x)) &= E\big(u \oplus E(x \oplus E(0))\big) \oplus E\big(x \oplus E(0)\big) \oplus E\big(0\big) \\ \Sigma'(\psi_u(y)) &= E\big(u \oplus d \oplus E(y \oplus E(0))\big) \\ \Theta'(\psi_u(y)) &= E\big(u \oplus d \oplus E(y \oplus E(0))\big) \oplus E\big(y \oplus E(0)\big) \oplus E\big(0\big) \end{aligned}$$

In particular, we observe

$$\begin{aligned} E(x \oplus E(0)) \oplus E(y \oplus E(0)) = d \Leftrightarrow \Sigma'(\phi_u(x)) &= \Sigma'(\psi_u(y)) \\ \Rightarrow \Theta'(\phi_u(x)) \oplus \Theta'(\psi_u(y)) &= d \\ \Rightarrow \mathrm{MAC}(\phi_u(x)) &= \mathrm{MAC}(\psi_u(y)). \end{aligned} \tag{8}$$

From this observation, we construct a birthday attack against `1kf9`. We build two lists:

$$L_0 = \Big\{\mathrm{MAC}(\phi_0(x)) \ : \ x < 2^{n/2}\Big\} \qquad L_1 = \Big\{\mathrm{MAC}(\psi_0(y)) \ : \ y < 2^{n/2}\Big\},$$

and we look for a match between the lists. We expect on average one pair to match randomly, and one pair to match because of (8). Moreover, when we have a collision candidate $L_0[x], L_1[y]$, we can verify whether it is a right pair by comparing $\mathrm{MAC}(x \parallel 1)$ and $\mathrm{MAC}(y \parallel d \oplus 1)$.

Therefore, we find a pair satisfying $\mathcal{R}(X,Y)$ with complexity $2^{n/2}$, and this leads to simple forgeries using (8). This contradicts the security proof of `1kf9` given in[8]. Note that this attack is still valid if we use different multiplications for the two branches in the finalization function.

6.2 Attacking `1kPMAC+`

The `1kPMAC+` mode uses the `fix` function for domain separation to build a single-key variant of `PMAC+`, as shown in Figure 8.

$$\begin{aligned} \Sigma'(M) &= \textstyle\bigoplus_{i=1}^{\ell} \tilde{E}_i(m_i) & \Sigma(M) &= \texttt{fix0}(\Sigma'(M)) \\ \Theta'(M) &= \textstyle\bigoplus_{i=1}^{\ell} 2^{\ell+1-i} \odot \tilde{E}_i(m_i) & \Theta(M) &= \texttt{fix1}(\Theta'(M)) \\ \mathrm{MAC}(M) &= E(\Sigma(M)) \oplus E(\Theta(M)) \end{aligned}$$

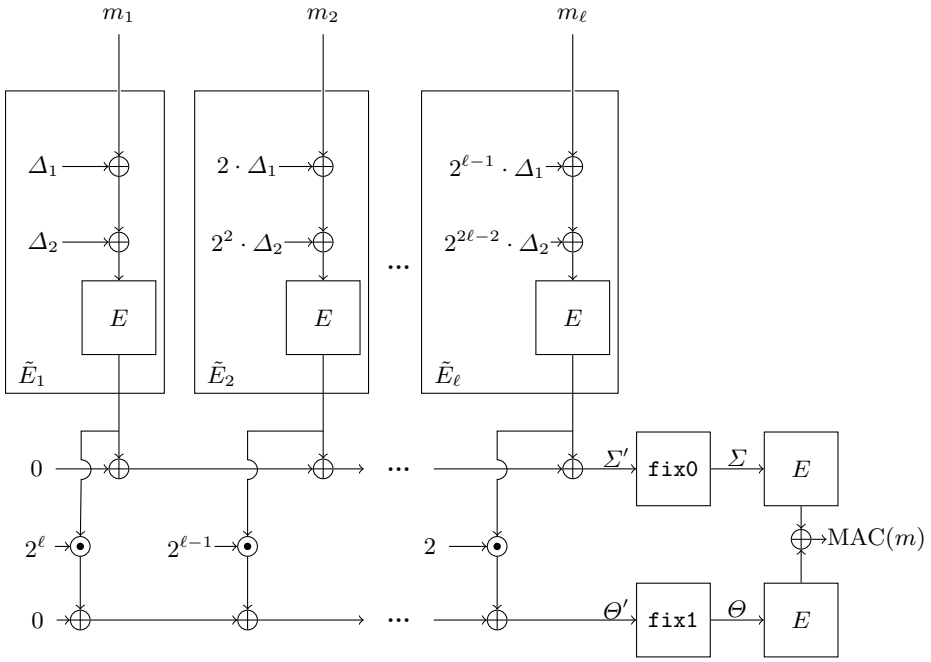


Fig. 8. Diagram for `1kPMAC+` with a ℓ-block message where $\Delta_1 = E(1)$ and $\Delta_2 = E(2)$.

Attack. Since the `fix` functions used in the finalization have collisions, we can build a variant of the attacks from Section 4 using differences in Σ' and/or Θ' that are absorbed by the `fix` functions. More precisely, we use the following relation $\mathcal{R}$ on quadruple of messages:

$$\mathcal{R}(X,Y,Z,T) := \begin{cases} \Sigma'(X) = \Sigma(Y)' \oplus 1 \\ \Theta'(Y) = \Theta(Z)' \oplus 1 \\ \Sigma'(Z) = \Sigma(T)' \oplus 1 \\ \Theta'(T) = \Theta(X)' \oplus 1 \end{cases}$$
$$\Rightarrow \mathrm{MAC}(X) \oplus \mathrm{MAC}(Y) \oplus \mathrm{MAC}(Z) \oplus \mathrm{MAC}(T) = 0.$$

We can find quadruple of messages satisfying $\mathcal{R}$ using a single message injection function:

$$\phi_u(i) = u \,\|\, i$$
$$X = \phi_u(x) = u \,\|\, x \quad Y = \psi_u(y) = u \,\|\, y \quad Z = \phi_u(z) = u \,\|\, z \quad T = \psi_u(t) = u \,\|\, t$$

Indeed we have

$$\mathrm{MAC}(\phi_u(i)) = E\Big(\mathtt{fix0}\big(\underbrace{\tilde{E}_1(u) \oplus \tilde{E}_2(x)}_{\Sigma'_u(i)}\big)\Big) \oplus E\Big(\mathtt{fix1}\big(\underbrace{4\tilde{E}_1(u) \oplus 2\tilde{E}_2(x)}_{\Theta'_u(i)}\big)\Big)$$

We observe that:

$$\mathcal{R}(x,y,z,t) \Leftrightarrow \begin{cases} \tilde{E}_2(x) = \tilde{E}_2(y) \oplus 1 \\ \tilde{E}_2(z) = \tilde{E}_2(t) \oplus 1 \\ 2\tilde{E}_2(x) = 2\tilde{E}_2(z) \oplus 1 \\ 2\tilde{E}_2(y) = 2\tilde{E}_2(t) \oplus 1 \end{cases}$$

$$\Leftrightarrow \begin{cases} \tilde{E}_2(x) \oplus \tilde{E}_2(y) \oplus \tilde{E}_2(z) \oplus \tilde{E}_2(t) = 0 \\ \tilde{E}_2(x) = \tilde{E}_2(y) \oplus 1 \\ \tilde{E}_2(x) = \tilde{E}_2(z) \oplus d \end{cases}$$

Therefore, $\mathcal{R}$ defines a $3n-$bit relation that is independent of the value u. This can be used for attacks in the same way as in the previous sections, using a single list

$$L = \left\{ \mathrm{MAC}(\phi_0(x)) \parallel \mathrm{MAC}(\phi_1(x)) \parallel \mathrm{MAC}(\phi_2(x)) \ : \ x < 2^{3n/4} \right\}$$

We can find a quadruple of four distinct values (x,y,z,t) such that $L[x] \oplus L[y] \oplus L[z] \oplus L[t] = 0$ with $\tilde{\mathcal{O}}(2^{3n/2})$ operations, using a memory of size $\mathcal{O}(2^{3n/4})$, and this easily leads to forgeries.

7 Conclusion

In this paper we have introduced a cryptanalysis technique to attack double-block-hash MACs using quadruples of messages. We show three variants of the technique, with attacks with $\mathcal{O}(2^{3n/4})$ queries against `SUM-ECBC`, `GCM-SIV2`, `PMAC+`, `LightMAC+`, `1kPMAC+` and `3kf9`. All these modes have a security proof up to $2^{2n/3}$ queries, but no attacks with fewer than 2^n queries were known before our work.

Our main attacks are in the information theoretic model, and an attacker would need more than 2^n operations to perform a forgery. On the other hand, we also have a variant of the attack against `SUM-ECBC` and `GCM-SIV2` with time complexity $\tilde{O}(2^{6n/7})$. This opens the path for attack with total complexity below 2^n for other double-block-hash MACs.

We believe that studying generic attacks is important in order to understand the security of these MACs, and is needed in addition to security proofs. In particular our results show that they do not reach full security, and we invalidate a recent proof for `LightMAC+`. However, there is still a gap between the $2^{2n/3}$ bound of the proofs, and our attacks with $\mathcal{O}(2^{3n/4})$ queries. Further work is needed to determine whether the attacks can be improved, or whether better proofs are possible.

Acknowledgement

Mridul Nandi is supported by R.C.Bose Centre for Cryptology and Security. Part of this work was supported by the French DGA.

A SageMath Implementation

In order to verify that the algorithm is correct, we have implemented the attack against SUM-ECBC with complexity $\tilde{\mathcal{O}}(2^{6n/7})$ given in Section 3.1 with SageMath:

```
xor  = lambda x, y: x.__xor__(y)
txor = lambda a,b: tuple(xor(u,v) for u,v in zip(a,b) )
def random_perm(n):
  pp = Permutations(n).random_element()
  return lambda x: pp(x+1)-1

def CBC(E,M):
  x = 0
  for m in M:
    x = E(x.__xor__(m))
  return x
def SUMECBC(E1,E2,E3,E4,M):
  a = E2(CBC(E1,M))
  b = E4(CBC(E3,M))
  return a.__xor__(b)

E1, E2, E3, E4 = (random_perm(2^21) for _ in range(4))
MAC = lambda x: SUMECBC(E1,E2,E3,E4,x)
print "Values to recover       | {0:06x} {1:06x}".format(
  xor(E1(0),E1(1)), xor(E3(0),E3(1)))

print "Generating data..."
L1,L2,L3,L4 = [], [], [], []
for i in range(2^12):
  if (i&0b000000000111 == 0): L1.append(i)
  if (i&0b000000111000 == 0): L2.append(i)
  if (i&0b000111000000 == 0): L3.append(i)
  if (i&0b111000000000 == 0): L4.append(i)
def macs(u,i):
  x = (0,0)
  for j in range(i,2^21,2^12):
    x = txor(x,(MAC([u,j]), MAC([u, xor(1,j)])))
  return (i,x)
L1 = [ macs(0,i) for i in L1 ]
L2 = [ macs(0,i) for i in L2 ]
L3 = [ macs(1,i) for i in L3 ]
L4 = [ macs(1,i) for i in L4 ]

print "Looking for quadruples..."
L13 = sorted((txor(a[1],b[1]),a[0],b[0]) for a in L1 for b in L3)
L24 = sorted((txor(a[1],b[1]),a[0],b[0]) for a in L2 for b in L4)
```

```
i,j = 0,0
while i<len(L13) and j<len(L24):
  if L13[i][0] == L24[j][0]:
    if L13[i] != L24[j]:
      print "{:06x} {:06x} {:06x} {:06x} | {:06x} {:06x}".format(
        L13[i][1], L13[i][2], L24[j][1], L24[j][2],
        xor(L13[i][1],L13[i][2]), xor(L13[i][1],L24[j][2]))
    if L13[i] < L24[j]:
      i+=1
    else:
      j+=1
  elif L13[i][0] < L24[j][0]:
    i+=1
  else:
    j+=1
```

References

1. An, J.H., Bellare, M.: Constructing VIL-MACs from FIL-MACs: Message authentication under weakened assumptions. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 252–269. Springer, Heidelberg (Aug 1999)
2. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Koblitz, N. (ed.) CRYPTO'96. LNCS, vol. 1109, pp. 1–15. Springer, Heidelberg (Aug 1996)
3. Bellare, M., Guérin, R., Rogaway, P.: XOR MACs: New methods for message authentication using finite pseudorandom functions. In: Coppersmith, D. (ed.) CRYPTO'95. LNCS, vol. 963, pp. 15–28. Springer, Heidelberg (Aug 1995)
4. Bellare, M., Kilian, J., Rogaway, P.: The security of the cipher block chaining message authentication code. Journal of Computer and System Sciences 61(3), 362–399 (2000)
5. Black, J., Rogaway, P.: A block-cipher mode of operation for parallelizable message authentication. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 384–397. Springer, Heidelberg (Apr / May 2002)
6. Chose, P., Joux, A., Mitton, M.: Fast correlation attacks: An algorithmic point of view. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 209–221. Springer, Heidelberg (Apr / May 2002)
7. Cogliati, B., Seurin, Y.: EWCDM: An efficient, beyond-birthday secure, nonce-misuse resistant MAC. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 121–149. Springer, Heidelberg (Aug 2016)
8. Datta, N., Dutta, A., Nandi, M., Paul, G., Zhang, L.: Building single-key beyond birthday bound message authentication code. Cryptology ePrint Archive, Report 2015/958 (2015), http://eprint.iacr.org/2015/958
9. Datta, N., Dutta, A., Nandi, M., Paul, G., Zhang, L.: Single key variant of PMAC_Plus. IACR Trans. Symm. Cryptol. 2017(4), 268–305 (2017)
10. Dinur, I., Leurent, G.: Improved generic attacks against hash-based MACs and HAIFA. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 149–168. Springer, Heidelberg (Aug 2014)

11. Dutta, A., Jha, A., Nandi, M.: Tight security analysis of EHtM MAC. IACR Trans. Symm. Cryptol. 2017(3), 130–150 (2017)
12. Ferguson, N.: Authentication weaknesses in GCM. Comment to NIST (2005), http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/CWC-GCM/Ferguson2.pdf
13. Computer data authentication. National Bureau of Standards, NIST FIPS PUB 113, U.S. Department of Commerce (1985)
14. Fuhr, T., Leurent, G., Suder, V.: Collision attacks against CAESAR candidates - forgery and key-recovery against AEZ and Marble. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, Part II. LNCS, vol. 9453, pp. 510–532. Springer, Heidelberg (Nov / Dec 2015)
15. Gilbert, E.N., MacWilliams, F.J., Sloane, N.J.: Codes which detect deception. Bell Labs Technical Journal 53(3), 405–424 (1974)
16. Guo, J., Peyrin, T., Sasaki, Y., Wang, L.: Updates on generic attacks against HMAC and NMAC. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 131–148. Springer, Heidelberg (Aug 2014)
17. Iwata, T.: New blockcipher modes of operation with beyond the birthday bound security. In: Robshaw, M.J.B. (ed.) FSE 2006. LNCS, vol. 4047, pp. 310–327. Springer, Heidelberg (Mar 2006)
18. Iwata, T., Kurosawa, K.: OMAC: One-key CBC MAC. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 129–153. Springer, Heidelberg (Feb 2003)
19. Iwata, T., Mennink, B., Vizár, D.: CENC is optimally secure. Cryptology ePrint Archive, Report 2016/1087 (2016), http://eprint.iacr.org/2016/1087
20. Iwata, T., Minematsu, K.: Stronger security variants of GCM-SIV. IACR Trans. Symm. Cryptol. 2016(1), 134–157 (2016), http://tosc.iacr.org/index.php/ToSC/article/view/539
21. Iwata, T., Minematsu, K., Peyrin, T., Seurin, Y.: ZMAC: A fast tweakable block cipher mode for highly secure message authentication. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 34–65. Springer, Heidelberg (Aug 2017)
22. Jaulmes, É., Joux, A., Valette, F.: On the security of randomized CBC-MAC beyond the birthday paradox limit: A new construction. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 237–251. Springer, Heidelberg (Feb 2002)
23. Joux, A.: Multicollisions in iterated hash functions. Application to cascaded constructions. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 306–316. Springer, Heidelberg (Aug 2004)
24. Knudsen, L.R., Mitchell, C.J.: Analysis of 3gpp-mac and two-key 3gpp-mac. Discrete Applied Mathematics 128(1), 181 – 191 (2003), http://www.sciencedirect.com/science/article/pii/S0166218X02004444, international Workshop on Coding and Cryptography (WCC2001).
25. Lee, C., Kim, J., Sung, J., Hong, S., Lee, S.: Forgery and key recovery attacks on PMAC and mitchell's TMAC variant. In: Batten, L.M., Safavi-Naini, R. (eds.) ACISP 06. LNCS, vol. 4058, pp. 421–431. Springer, Heidelberg (Jul 2006)
26. Leurent, G., Peyrin, T., Wang, L.: New generic attacks against hash-based MACs. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 1–20. Springer, Heidelberg (Dec 2013)
27. List, E., Nandi, M.: Revisiting full-PRF-secure PMAC and using it for beyond-birthday authenticated encryption. In: Handschuh, H. (ed.) CT-RSA 2017. LNCS, vol. 10159, pp. 258–274. Springer, Heidelberg (Feb 2017)
28. List, E., Nandi, M.: ZMAC$^+$ – an efficient variable-output-length variant of ZMAC. IACR Trans. Symm. Cryptol. 2017(4), 306–325 (2017)

29. Luykx, A., Preneel, B., Tischhauser, E., Yasuda, K.: A MAC mode for lightweight block ciphers. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 43–59. Springer, Heidelberg (Mar 2016)
30. Mennink, B., Neves, S.: Encrypted davies-meyer and its dual: Towards optimal security using mirror theory. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 556–583. Springer, Heidelberg (Aug 2017)
31. Minematsu, K.: How to thwart birthday attacks against MACs via small randomness. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 230–249. Springer, Heidelberg (Feb 2010)
32. Naito, Y.: Full PRF-secure message authentication code based on tweakable block cipher. In: Au, M.H., Miyaji, A. (eds.) ProvSec 2015. LNCS, vol. 9451, pp. 167–182. Springer, Heidelberg (Nov 2015)
33. Naito, Y.: Blockcipher-based MACs: Beyond the birthday bound without message length. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part III. LNCS, vol. 10626, pp. 446–470. Springer, Heidelberg (Dec 2017)
34. Naito, Y.: Improved security bound of LightMAC_Plus and its single-key variant. In: Smart, N.P. (ed.) CT-RSA 2018. LNCS, vol. 10808, pp. 300–318. Springer, Heidelberg (Apr 2018)
35. Nikolic, I., Sasaki, Y.: Refinements of the k-tree algorithm for the generalized birthday problem. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, Part II. LNCS, vol. 9453, pp. 683–703. Springer, Heidelberg (Nov / Dec 2015)
36. Peyrin, T., Wang, L.: Generic universal forgery attack on iterative hash-based MACs. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 147–164. Springer, Heidelberg (May 2014)
37. Preneel, B., van Oorschot, P.C.: MDx-MAC and building fast MACs from hash functions. In: Coppersmith, D. (ed.) CRYPTO'95. LNCS, vol. 963, pp. 1–14. Springer, Heidelberg (Aug 1995)
38. Preneel, B., van Oorschot, P.C.: On the security of two MAC algorithms. In: Maurer, U.M. (ed.) EUROCRYPT'96. LNCS, vol. 1070, pp. 19–32. Springer, Heidelberg (May 1996)
39. Rogaway, P.: Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 16–31. Springer, Heidelberg (Dec 2004)
40. Wagner, D.: A generalized birthday problem. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 288–303. Springer, Heidelberg (Aug 2002)
41. Wegman, M.N., Carter, L.: New hash functions and their use in authentication and set equality. Journal of Computer and System Sciences 22, 265–279 (1981)
42. Yasuda, K.: The sum of CBC MACs is a secure PRF. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 366–381. Springer, Heidelberg (Mar 2010)
43. Yasuda, K.: A new variant of PMAC: Beyond the birthday bound. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 596–609. Springer, Heidelberg (Aug 2011)
44. Zhang, L., Wu, W., Sui, H., Wang, P.: 3kf9: Enhancing 3GPP-MAC beyond the birthday bound. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 296–312. Springer, Heidelberg (Dec 2012)