Key-Recovery Attacks on Universal Hash Function based MAC Algorithms*

Helena Handschuh[1] and Bart Preneel[2,3]

[1] Spansion, 105 rue Anatole France 92684 Levallois-Perret Cedex, France
helena.handschuh@spansion.com
[2] Katholieke Universiteit Leuven, Dept. Electrical Engineering-ESAT/COSIC, Kasteelpark Arenberg 10, bus 2446, B-3001 Leuven, Belgium
bart.preneel@esat.kuleuven.be
[3] IBBT, Van Crommenlaan, B-9000 Gent

Abstract. This paper discusses key recovery and universal forgery attacks on several MAC algorithms based on universal hash functions. The attacks use a substantial number of verification queries but eventually allow for universal forgeries instead of existential or multiple forgeries. This means that the security of the algorithms completely collapses once a few forgeries are found. Some of these attacks start off by exploiting a weak key property, but turn out to become full-fledged divide and conquer attacks because of the specific structure of the universal hash functions considered. Partial information on a secret key can be exploited too, in the sense that it renders some key recovery attacks practical as soon as a few key bits are known. These results show that while universal hash functions offer provable security, high speeds and parallelism, their simple combinatorial properties make them less robust than conventional message authentication primitives.

1 Introduction

Message Authentication Code (MAC) algorithms are symmetric cryptographic primitives that allow senders and receivers who share a common secret key to make sure the contents of a transmitted message has not been tampered with. Three main types of constructions for MAC algorithms can be found in the literature: constructions based on block ciphers, based on hash functions and based on universal hash functions. In this paper we focus on the third class. The interesting feature of these MAC algorithms is that they are secure against an opponent with unlimited computing power. Simmons performed seminal work in this area [35]. The most widely used schemes are constructed following the Wegman-Carter paradigm [37]: first, the input message is hashed to a short digest using a universal hash function indexed by a secret key. Such a hash function has the property that for any two distinct inputs the probability over

* This work was partially funded by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT and by the Belgian Government through the IUAP Programme under contract P6/26 BCRYPT.

all keys that two inputs have a specific difference is small. Then, the resulting hash value is encrypted by adding a one-time key. This approach is provably secure in the information theoretic setting. Brassard proposed a construction of a computationally secure MAC algorithm from universal hash functions by replacing the one-time pad by the output of a pseudo-random function applied to a nonce [11]. Later work suggests to apply a pseudo-random function directly to the hash result. A second optimization is to also derive the key for the universal hash function from a short key; if the universal hash function key is large (it can be several Kbytes long), this would still be too inefficient; instead one reuses this key for multiple messages. This paper will show that this opens the way for new attacks. A large number of computationally secure MAC algorithms derived from universal hash functions have been proposed following this model. Such MAC algorithms and universal hash functions include UMAC [9], MMH [17], NMH [17], Square Hash [15], Poly1305-AES [6], CWC [25], GCM/GMAC [30, 31], and a recent polynomial variant by Bernstein [7]. They are seen to be attractive because of their high speed and the simple constructions and security analysis.

Our contribution. This paper explores the implications of the reuse of key material in constructions of MAC algorithms based on universal hash functions as suggested by Wegman and Carter [37]; our work leads to improved forgery attacks and shortcut attacks that recover secret keys or parts thereof with the same complexity as a forgery attack. From a provable security viewpoint, it may seem desirable that forgery is equivalent to key recovery (since it implies that a weaker forgery attack can be reduced to the most difficult attack, namely a key recovery attack). However, key recovery allows for an arbitrary number of selective forgeries, hence its practical implications are much more serious. While for some applications it may be acceptable that a MAC forgery occurs from time to time, being able to recover the secret key as soon as the forgery bound is reached, is clearly not. It is important to stress that our results do not violate the proven security bounds on forgery attacks. Key recovery attacks are only a problem because MAC keys are being reused, which was not the case in the initial approach for authentication based on universal hash functions.

As their name suggests, universal hash functions guarantee good behavior of the hash function for *any* input pair; however, this refers to an average behavior over all keys and does not guarantee that each key yields a hash function with a uniform output distribution. For some schemes we identify rather large classes of weak keys that allow to easily forge authentication tags by swapping two blocks or by assigning specific values to some message blocks. The use of a weak key can typically be detected with a single text/MAC pair: it is sufficient to modify the text and submit a verification query. In principle the parties could check for the presence of weak keys, but in some cases this will substantially increase the complexity of the key generation procedure since a large number of combinations need to be avoided.

While the forgery probability of these schemes can be bounded, we show that in several MAC algorithms a small number of forgeries leads to if not total at least partial key recovery, which in turn allows us to create an arbitrary number of forgeries – unlike conventional MAC algorithms such as CBC-MAC [18, 32] the security of MAC algorithms based on universal hash functions collapses once a few forgeries are found; sometimes even a single forgery suffices.

For some constructions, we present enhanced key recovery attacks. We guess part of the key and try to confirm this guess with a MAC verification query. In schemes with large keys consisting of many words (say of 32 bits each), this allows for an efficient word by word key recovery. Some of our key recovery attacks can take into account partial information we may have obtained on a key, for example if sender and verifier read out a few bytes of the key over the phone to make sure they are using the right key or when side-channel attacks reveal some key bits or relations among key bits or their Hamming weight.

A final class of attacks, described for the purpose of completeness, exploits birthday attacks on special types of messages. If the MAC computation is stateful, that is, if the MAC generation algorithm depends on a nonce (a random number or a counter), these attacks require reuse of nonces by the sender (counter misuse), which is typically not allowed by the security model. However, one should expect that in some cases attackers can 'reset' senders (think of a bug in the code or a fault injection that make the software used by the sender crash). It is an unfortunate property of these MAC algorithms that a limited number of sender resets results in serious security weaknesses.

Most of our attacks however do *not* require nonce reuse at all, since we mostly consider a very simple person-in-the-middle scenario: the sender chooses a unique nonce and creates a text/MAC pair. The attacker modifies the text and submits a verification query *without changing the tag or the nonce*. The idea behind our attacks is that messages for which the resulting hash values collide by construction can be substituted without knowing (or even re-using) the one-time value the hash result was encrypted with. This scenario clearly fits within the security model of MAC algorithms (see e.g., Bellare et al. [4]).

Some of the results may seem rather straightforward and a few of these observations have been made earlier on some schemes. However, designers of universal hash functions typically do not emphasize these weaknesses. To the best of our knowledge, this paper is the first work that systematically investigates how robust or brittle a broad series of universal hash functions is when some of the above observations are applied. It also shows how a simple weakness can easily be developed to a partial key recovery attack with devastating consequences. In particular, we show new findings on the following universal hash functions and MAC algorithms based thereupon: polynomial hash [6, 13, 8, 22, 25], MMH [17], Square Hash [15], NH/UMAC [9] and its variants VMAC [28] and WH/WMAC [23]. Due to space limitations, the results on Johansson's bucket hashing with a short key size [20] only appear in the long version of this paper.

Related work. McGrew and Fluhrer have observed in [29] that once a single forgery has been performed, additional forgeries become easier; more specifically, the forgery probability for MAC algorithms such as CBC-MAC and HMAC increases cubically with the number of known text-MAC pairs, while for universal hash functions the forgery probability increases only quadratically.

Black and Cochran have analyzed in [10] what happens after a first collision is observed in the output of the MAC algorithm; they show that subsequent forgeries or reforgeability becomes easier than expected; for two cases they also present a key recovery attack. They propose a solution to these reforgeability attacks in the form of the WMAC construction [10]; however, we will show in this paper that this construction does not offer any protection against most of our key recovery attacks.

There is a broad literature on key recovery attacks on MAC algorithms (e.g., [34]). The techniques used in this paper are based on classical divide-and-conquer techniques. Our attacks on GCM/GMAC [30, 31] extend earlier work by Ferguson [16] and Joux [21] by considering the case where the one-time pad (more precisely the addition of the output of a pseudo-random function) is replaced by the application of a pseudo-random function.

Organization of the paper. In Sect. 2 we provide some background on MAC algorithms, information theoretic authentication and universal hash functions. Section 3 describes our attacks and discusses a number of attacks on different schemes in detail. In Sect. 4 we summarize our results and present some open problems and directions for future work.

2 Background

2.1 MAC Algorithm

A MAC algorithm consists of three algorithms: 1) a key generation algorithm, that is, a randomized algorithm that generates a κ-bit key k; 2) a MAC generation algorithm, that on input a text x and a key k generates an m-bit tag (this algorithm can be randomized and/or can be stateful); 3) a MAC verification algorithm, that on input a text and a tag generates an answer true or false (1/0) (this algorithm is deterministic and typically stateless). The security of a MAC algorithm can be formalized in concrete complexity following Bellare et al. [5] (see also [33]).

Definition 1. *A MAC algorithm is* $(\epsilon, t, q, q', q'', L)$ *secure if, an adversary who does not know* k*, and*

- *can spend time* t *(operations);*
- *can obtain the MAC for* q *chosen texts;*
- *can observe the MAC for* q' *known texts; and*
- *can obtain the result of* q'' *verification queries on text-MAC pairs of his choice.*

(each text of length L*), cannot produce an existential forgery with probability of success larger than* ϵ*.*

Here known text-MAC pairs are generated according to a distribution chosen by the sender, while chosen text-MAC pairs are generated according to a distribution specified by an adversary.

There are two generic attacks on MAC algorithms: finding the key by exhaustive search, which requires κ/m known text-MAC pairs and on average $2^{\kappa-1}$ (off-line) MAC evaluations and guessing the MAC value, which requires on average $\min(2^{m-1}, 2^{\kappa-1})$ MAC verifications. The second attack can be detected easily by the large number of wrong MAC verifications. Iterated MAC algorithms with an n-bit internal memory succumb to a birthday forgery attack that requires $2^{n/2}$ known text-MAC pairs and $\min(2^{n/2}, 2^{n-m})$ chosen text-MAC pairs [24, 34].

Typical MAC algorithms used in practice are CBC-MAC (the variants EMAC [32, 18] and CMAC [19] work for arbitrary length messages) and HMAC [3, 2].

2.2 Information theoretic authentication

Information theoretic authentication was developed in the 1970s by Simmons [35] and Carter and Wegman [12, 37]. It has several very attractive features. First and foremost, its security is not based on any complexity theoretic assumption and holds against opponents with unlimited computing power. In the last decade researchers have attempted to fine tune the speed of these constructions; the current state of the art is that they are up to 15 times faster than CBC-MAC based on AES or HMAC based on SHA-1. Moreover, if properly designed, these functions can be parallelizable and incremental; the latter property means that if a local change is made to a large message, the MAC value can be updated in time that depends on the size of the local update rather than on the overall message length. Information theoretic authentication also has serious disadvantages: as for the one-time pad, it is required that the key is used only once. Moreover, in order to get extreme performance (below 2 cycles/byte in software), very large keys are required which increases key storage costs and limits key agility. Finally, note that Simmons' theory shows that the security level in bits against forgery is at most half the key size in bits. This paper will demonstrate that these constructions have another disadvantage; however, this disadvantage will only become apparent if the information theoretic scheme is replaced by a computationally secure scheme with comparable efficiency.

One of the most elegant examples of an information theoretic authentication scheme is the polynomial construction (see e.g., [8, 13]). This construction uses two n-bit keys k and k', and operates on messages of bitlength $\ell = t \cdot n$. It starts by splitting the input message x into t blocks of bitlength n denoted x_1 through x_t. The x_i, k and k' are represented as elements of $\mathrm{GF}(2^n)$. The authentication function $g_{k,k'}(x)$ can then be described as follows:

$$g_{k,k'}(x) = k' + \sum_{i=1}^{t} x_i \cdot k^i .$$

The probability of sending an acceptable tag without having observed a text/tag pair (the impersonation probability) equals 2^{-n}, while the probability of forging a text/tag pair after having observed one pair (the substitution probability) is equal to $(\ell/n)/2^n = t/2^n$. It has been pointed out that the value of k can be reused for multiple messages, as long as the value of k' is used only once (e.g., [13]).

2.3 MAC algorithms based on universal hashing

Following Carter and Wegman [12], a universal hash function is a family of functions indexed by a parameter called the key with the following property: for all distinct inputs, the probability over all keys that they collide is small.

Definition 2. *Let $g_k : A \longrightarrow B$ be a family of functions with $a = |A|$ and $b = |B|$. Let ϵ be any positive real number. Then g_k is an* **ϵ-almost universal** *class (or ϵ-AU class) $\mathcal{G}$ of hash functions if*

$$\forall\, x, x' \neq x \in A :\ \Pr_k \{g_k(x) = g_k(x')\} \leq \epsilon\,.$$

These functions can only be used for message authentication if the output is processed using another function. If one wants to use a universal hash function directly for message authentication, a stronger property is required, namely the hash function needs to be strongly universal [12, 37]. This concept has later on been generalized by Krawczyk [26].

Definition 3. *Let $B, \star$ be an Abelian group. Then g_k is an* **ϵ-almost $\star$ universal** *class (or ϵ-A$\star$U class) of hash functions if*

$$\forall\, x, x' \neq x \in A \text{ and } \forall \Delta \in B :\ \Pr_k \{g_k(x) = g_k(x') \star \Delta\} \leq \epsilon\,.$$

Many simple mathematical operations, such as polynomial evaluation, matrix-vector product and inner products, yield ϵ-AU or ϵ-A$\star$U hash functions.

A MAC algorithm based on universal hash functions consists of two building blocks: an efficient keyed compression function that reduces long inputs to a fixed length and a method to process the short hash result and an output transformation. In practical constructions, the encryption with the one-time pad (addition of k') is typically replaced by applying a pseudo-random function with secret key k'. In this case one obtains computational rather than unconditional security. Informally, a pseudo-random function family is a function that a computationally limited adversary cannot distinguish with probability substantially better than 1/2 from a function chosen uniformly at random from all functions with the same range and domain.

The following compression functions have been proposed in practice. We consider an input that consists of t blocks of fixed length; if necessary, t will be assumed to be even. A simple ϵ-AU hash function is the polynomial evaluation function over a finite field:

– $g_k(x) = \sum_{i=0}^{t} x_i \cdot k^i\,,\ \ x_i, k \in \mathrm{GF}(2^n)$ or $\mathrm{GF}(p)$.

The following functions are ϵ-A$\star$U for $\star$ equal to addition modulo 2, 2^n or p (this is clear from the context):

– polynomial [30, 31, 6]: $g_k(x) = \sum_{i=1}^{t} x_i \cdot k^i$, $x_i, k \in \mathrm{GF}(2^n)$ or $\mathrm{GF}(p)$;
– MMH [17]: $g_k(x) = \left(\left(\left(\sum_{i=1}^{t} x_i \cdot k_i\right) \bmod 2^{64}\right) \bmod p\right) \bmod 2^{32}$, $x_i, k_i \in \mathbf{Z}_{2^{32}}$ and $p = 2^{32} + 15$;
– Square Hash [15]: $g_k(x) = \sum_{i=1}^{t} (x_i + k_i)^2 \bmod p$, $x_i, k_i \in \mathbf{Z}_{2^w}$;
– NMH [37, 17]: $g_k(x) = \left(\sum_{i=1}^{t/2} (x_{2i-1} + k_{2i-1}) \cdot (x_{2i} + k_{2i})\right) \bmod p$, $x_i, k_i \in \mathbf{Z}_{2^{32}}$ and $p = 2^{32} + 15$;
– NH [9]: $g_k(x) = \left(\sum_{i=1}^{t/2} ((x_{2i-1} + k_{2i-1}) \bmod 2^w) \cdot ((x_{2i} + k_{2i}) \bmod 2^w)\right) \bmod 2^{2w}$, x_i, $k_i \in \mathbf{Z}_{2^w}$;
– WH [23] $g_k(x) = \left(\sum_{i=1}^{t/2} (x_{2i-1} + k_{2i-1}) \cdot (x_{2i} + k_{2i}) x^{(t/2-i)w}\right) \bmod p(x)$, x_i, $k_i \in \mathrm{GF}(2^w)$ (polynomial arithmetic).

Note that one can also cascade multiple ϵ-AU hash functions followed by an ϵ-A$\star$U hash function to obtain more efficient constructions (see Stinson [36]).

For the output transformation, the options listed below can be considered; this list has been inspired by the slides of [6]. Here $f_{k'}()$ denotes a pseudo-random function family indexed by the key k'. The first option results in a stateless MAC algorithm, while the other two assume that the sender keeps the state with the counter n (which is called the nonce hereafter). The last two options offer a better security level but one needs to guarantee that n is not reused during the MAC generation. Nonce reuse during the MAC verification is typically allowed.

Option 1: $\mathrm{MAC}_{k||k'}(x) = f_{k'}(g_k(x))$ with g ϵ-AU.
Option 2: $\mathrm{MAC}_{k||k'}(x) = f_{k'}(n) \star g_k(x)$ with g ϵ-A$\star$U;
Option 3: $\mathrm{MAC}_{k||k'}(x) = f_{k'}(n||g_k(x))$ with g ϵ-AU; this variant needs a larger input of f.

A security analysis of Option 3 (under the name WMAC[4]) is provided by Black and Cochran in [10]. Due to space restrictions we omit an analysis of the randomized message preprocessing of Dodis and Pietrzak [14].

The MAC algorithms based on universal hash functions considered in standards are UMAC [27] and GCM [31]; both follow Option 2. ISO/IEC JTC1/SC27 is currently developing an international standard on MAC algorithms based on universal hash functions (part 3 of IS 9797).

3 Security Results

In this central section of the paper we present new results on the security of universal hash function based MAC algorithms. We present a detailed analysis of the vulnerability of several MAC algorithms with respect to three different scenarios. First we briefly describe each of the algorithms and immediately identify classes of weak keys for each of them. Subsequently we describe more powerful universal forgery and key recovery attacks based on partial key information leakage and the divide-and-conquer principle. Next we describe a key recovery attack on polynomial hash functions using the birthday paradox.

[4] Unfortunately this name has already been given by Kaps et al. to a completely different MAC algorithm one year earlier [23], cf. Sect. 3.1.

3.1 Weak keys

In symmetric cryptology, a class of keys is called a weak key class if for the members of that class the algorithm behaves in an unexpected way *and* if it is easy to detect whether a particular unknown key belongs to this class. For a MAC algorithm, the unexpected behavior can be that the forgery probability for this key is substantially larger than average. Moreover, if a weak key class is of size C, one requires that identifying that a key belongs to this class requires testing fewer than C keys by exhaustive search and fewer than C verification queries.

The security proofs for universal hash functions bound the average forgery probability over all keys; this does not cause any problem if the key is used only once since for most applications a bound on the average rather than the worst case forgery probability is sufficient. However, if the key of the universal hash function is reused (which is the case in many current constructions), the fact that the weak keys are easy to recover can have dramatic implications, as key recovery allows for arbitrary forgeries.

Polynomial hash functions. We first study the ϵ-A$\oplus$U polynomial hash function $g_k(x) = \sum_{i=1}^{t} x_i \cdot k^i$ with $k, x_i \in \mathrm{GF}(2^n)$. The forgery probability of this scheme is equal to $t/2^n$ (cf. Sect. 2.2). This function forms the core of the GMAC (Galois MAC) and GCM (Galois/Counter Mode of operation) constructions for a MAC algorithm and for authenticated encryption by McGrew and Viega [30]; both have been standardized by NIST [31]. The GCM construction uses Option 2 of Sect. 2.3 with the pseudo-random value generated by a block cipher in counter (CTR) mode. It allows to truncate the output by selecting the leftmost τ bits.

One can also consider polynomials evaluated over $\mathrm{GF}(p)$ to take advantage of fast multiplication hardware available on current processors. The fastest scheme today is Poly1305-AES of Bernstein [6] that uses $p = 2^{130} - 5$.

Clearly $k = 0$ is a weak key for these polynomial hash functions. In this case (which appears with probability 2^{-n} respectively $1/p$) all messages map to 0 under g_k and all messages have the same valid tag, allowing for trivial forgery attacks: an attacker can take any valid text/MAC pair and substitute the text by any text of her choice (that is, the forgery probability is 1 independent of the output transformation). As 2^n and p are typically very large, this is not a realistic threat.

MMH. This inner product construction was introduced by Halevi and Krawczyk in 1997 [17]. It is an ϵ-A+U with $\epsilon = 1.5/2^{30}$.

Next we describe two weak key classes for MMH. A first class consists of the keys for which any k_i value is equal to 0. We present a more general description by considering a universal hash function family $g_k(x)$ with k consisting of t elements k_i of the ring R and x consisting of t elements x_i of the ring R.

Proposition 1 (type I). *If $g_k(x)$ is of the form $z_0\left(\sum_{i=1}^{t} z_1(k_i, x_i)\right)$ where $z_1(k_i, x_i)$ is a multiple of k_i and $z_0()$ is an arbitrary function, then any key k*

for which at least one $k_i = 0$ *belongs to a weak key class. The fraction of weak keys is equal to* $1 - (1 - 1/|R|)^t \approx t/|R|$*. Membership in this class can be tested with* t *MAC queries and* t *verification queries.*

The proposition assumes that one asks for a text/MAC pair (if applicable with a given nonce), one modifies x_i, which implies that the MAC value does not change if $k_i = 0$, and submits a verification query with the modified x_i but the original MAC value. Note that one could reduce the number of text/MAC pairs to a single one by modifying always the same text/MAC pair. If a nonce is used, this requires the assumption that the verifier does not check repeating nonces (which is a standard assumption in MAC algorithms), but our attacks also apply (with a higher complexity) if the verifier does check for repeating nonces. Hence this attack is equally effective whether Option 1, 2, or 3 is used. This comment also applies to the other weak key classes identified below.

A second weak key class is based on the summation and exploits two equal subkeys:

Proposition 2 (type II). *If* $g_k(x)$ *is of the form* $z_0\left(\sum_{i=1}^{t} z_2(k_i, x_i)\right)$ *for any functions* $z_2(k_i, x_i)$ *and* $z_0()$*, then any key* k *for which there exist* i *and* j *such that* $k_i = k_j$ *belongs to a weak key class. The fraction of weak keys is equal to* $R^t - R!/(R-t)! \approx t(t-1)/(2|R|)$ *(for* t *small). Membership in this class can be tested with* $t(t-1)/2$ *MAC queries and* $t(t-1)/2$ *verification queries.*

This class can be further generalized: if $k_i = \alpha \cdot k_j$ (over the integers), this can be detected as follows: one asks for a message, substitutes the blocks x_i and x_j by $x'_i = x_j/\alpha$ and $x'_j = \alpha \cdot x_i$ and makes a verification query (this requires that α divides x_j and x_i is smaller than $2^{32}/\alpha$). Type II weak keys correspond to $\alpha = 1$; they can be identified by swapping two message blocks. The probability that two key words satisfy such a relation for a fixed value of α is equal to $1/(\alpha \cdot 2^{32})$.

Square Hash. This construction was introduced by Etzel, Patel and Ramzan [15] in 1999. Proposition 2 also results in weak keys for Square Hash.

NMH, NH and WH. The NMH construction by Wegman and Carter is a variant of MMH described in the same article [17]. It consists of an inner product of sums of message and key words. NMH requires a large key, but yields the fastest throughput for long messages on general processors as it can make use of multimedia instructions. NMH* is a variant of NMH in which the inner addition is done modulo 2^w for efficiency reasons, and the result of the modulo p operation is further reduced modulo 2^w. Typical choices are $w = 32$ and $p = 2^{32} + 15$.

NH is a yet another variant of NMH that uses additional modular reductions to improve the performance (this reduction corresponds to ignoring carry bits). It forms the crucial building block of the UMAC and VMAC algorithms. UMAC [9, 27] was introduced by Black, Halevi, Krawczyk, Krovetz and Rogaway in 1999. UMAC first applies a universal hash function called UHASH to the message in order to derive a shorter fixed-length hash value. In order to obtain very high efficiency, UHASH is constructed of three layers, the first of which is NH, the second one is a polynomial hash function over GF(p) and the third one an MMH construction. In order to obtain a MAC algorithm, Option 2 is selected, similar to the GCM and Poly1305-AES constructions. VMAC is a variant of UMAC that is optimized for 64-bit arithmetic [28]. WH is a variant of NH that replaces elements of $\mathbf{Z}_{2^w}$ by polynomials (elements of GF(2^w)) to reduce the cost of hardware implementations; the resulting MAC is called WMAC [23].

This class of hash functions has weak keys that are based on symmetry:

Proposition 3 (type III). *If* $g_k(x)$ *is of the form* $z_0\left(\sum_{i=1}^{t/2} z_3(k_{2i-1}, x_{2i-1}) \cdot z_3(k_{2i}, x_{2i})\right)$ *for any functions* $z_3(k_i, x_i)$ *and* $z_0()$*, then any key* k *for which there exists an* i *such that* $k_{2i-1} = k_{2i}$ *belongs to a weak key class. The fraction of weak keys is equal to* $1 - (1 - 1/|R|)^{t/2} \approx t/2|R|$ *(for* t *small). Membership in this class can be tested with* $t/2$ *MAC queries and* $t/2$ *verification queries.*

One can also apply Proposition 2 to the $t/2$ terms in the sum.

We conclude this section with a simple numerical example to illustrate the impact of these weak keys. If $t = 256$ and $|R| = 2^{32}$, the keys are 8192 bits long and the fraction of weak keys for Proposition 1, 2, and 3 is equal to 2^{-24}, 2^{-17}, and 2^{-25} respectively; membership in this class can be verified with 256, 32 640, and 128 queries.

3.2 Divide and conquer attacks and partial key information leakage

In this section we show how the existence of these weak key classes lead us to discover efficient key recovery attacks and how partial information on the key can be exploited to further speed up these attacks. The techniques are based on the construction of inner collisions in the universal hash function (following [34]) which can be confirmed with a single message substitution and a single verification query.

Polynomial hash. We can verify a guess for part of the key (namely k), even if we do not know the key k' to the pseudo-random function. Assume for example that $t \geq 2$, $k \neq 0$ and $x_i = 0$, for $3 \leq i \leq t$. Then for a given x_1 and x_2 and a guess for k, we can choose any x'_2 and compute $x'_1 = x_1 + (x_2 - x'_2) \cdot k$. If the guess for k is correct, the message $x'_1||x'_2||0||\ldots$ will be accepted as a valid message for the same tag as message $x_1||x_2||0||\ldots$. Hence if this message is accepted, with high probability the guess for k was correct. It may also be that the guess for k was wrong but that due to a collision in the pseudo-random function $f_{k'}(.)$ the verification worked. This case can be ruled out by repeating the attack with a different message. Note that the attacker constructs here messages based on a text/MAC pair with a nonce selected by the sender, and the receiver uses each nonce only once. The GCM mode precludes this divide-and-conquer attack by deriving k from k'. In SNOW 3G [1] the key k is used only once, hence if the receiver checks for nonce reuse a correct guess for k can no longer be used.

Extensions of the Joux attacks on GCM variants. Joux presents an elegant attack on GCM which uses Option 2 with nonce reuse by the sender (the attack is described in Annex A). Here we show that we can extend his attack to Option 1 (even if classical encryption in ECB mode is used as suggested by Joux) or Option 3 (but without nonce reuse); this result shows that some of these weaknesses are inherent to polynomial hashing over $GF(2^n)$ and not to the use of the one-time pad or CTR mode as in GCM. In this case we have no access to the individual bits of $g_k(x) \oplus g_k(x')$ for two distinct messages x and x', but we can only test equality between the complete n-bit strings $g_k(x)$ and $g_k(x')$. We show that in this case a forgery also leads to key recovery, that is, recovering a key k requires an expected number of $\log_2(t) + 2^n/t$ verification queries. The attack goes as follows:

1. obtain a text x and the corresponding MAC value;
2. choose x' such that the polynomial with coefficients from $x - x'$ has t distinct roots (they should also be distinct from all previous roots used in this algorithm);
3. perform a MAC verification query for x'; if the result is incorrect, go to step 1;
4. after $2^n/t$ trials on average, you expect a correct MAC value and then you know that k is one of t values, that is, one of the at most t roots of a polynomial of step 2;
5. perform another $\log_2(t)$ MAC verification queries (binary search) to identify which one of the t roots is equal to k.

This attack is very similar to the key recovery attack by Black and Cochran [10]; the main difference is that [10] first assumes a forgery and then identifies the key, while here the forgery algorithm is optimized in function of the key recovery. We make the following comments:

- Exceptionally, this attack would also work with high probability if k is changed every time as for SNOW 3G [1] (in this case Step 5 would require nonce reuse by the receiver). The attack would then only be useful if the receiver could be convinced to reuse a key.
- If $n = 128$ as for GCM, this attack is clearly impractical. Even if $t = 2^{64}$ this would still require the verification of a large number of messages of length 2^{64}.
- The expected number of multiplications performed by the verification oracle is about $t \cdot 2^n/t = 2^n$, hence one may conclude that this attack is not better than brute force key search. First, it should be pointed out that this is a divide-and-conquer attack that applies independently of the key k' of the pseudo-random function. Second, this method may help to overcome limitations set on the number of incorrect MAC verifications, as it tests as many candidate key values as possible with a single verification query.
- It is straightforward to take into account in this attack any information one may have on the key k, such as the value of the first three bytes, the sum of any bits etc. Consider $n = 80$ and $t = 2^{18}$ and assume that we have an

oracle that gives us $s = 24$ key bits. Then the attack requires 2^{38} text-MAC verifications to find the remaining 56 key bits. The same observation also applies to the remainder of this section, since the attacker has to solve only very simple equations in the words of the key.

MMH. For the case of MMH, assume that one knows a multiplicative relation between two key words, say $k_i = (\alpha \cdot k_j) \bmod p$ for an $\alpha \in \mathbf{Z}_p$. Consider w.l.o.g. messages for which only x_i and x_j are non-zero. Then if one replaces (x_i, x_j) by (x'_i, x'_j) the hash value will be unchanged provided that $x'_j = (x_j + (x_i - x'_i) \cdot \alpha) \bmod p$. We require that the $\bmod 2^{64}$ operation can be ignored, which holds if $x_i \cdot k_i + x_j \cdot k_j < 2^{64}$ and $x'_i \cdot k_i + x'_j \cdot k_j < 2^{64}$. One way to ensure that the first inequality is met is by choosing $x_i, x_j < 2^{31}$; for the second inequality one chooses $x'_i < 2^{31}$. Then $x'_j < 2^{31}$ with probability approximately $1/2$, hence it is easy to find tuples (x_i, x_j) and (x'_i, x'_j) for which these equations hold. Note that for every value of α one can precompute solutions without knowing the key. This implies that a single verification query after a message substitution allows to verify a guess for α. Overall this means that 32 bits of information on the key can be found with approximately 2^{32} verification queries; subsequently arbitrary forgeries are possible. Recovery of the t key words requires $t \cdot 2^{32}$ verification queries.

Square Hash. Square hash has the weakness that one can verify a guess for any key word k_i of w bits with a single verification query: it suffices to modify x_i into $x'_i = (-2k_i - x_i) \bmod p$. This follows immediately from the fact that $(-x)^2 \equiv x^2 \bmod p$. This attack requires approximately p MAC verifications to find a single key word and $t \cdot p$ MAC verifications to identify the correct key (a divide-and-conquer approach). Note that this attack could be precluded by ensuring that the hash function is restricted to input values less than $p/2$. This attack is simpler than the collision-based key recovery attack of Black and Cochran [10].

NMH, NH and WH. The following attack applies to all these constructions: it is sufficient to find a message for which one factor becomes 0 (e.g., for NMH $(x_{2i-1} + k_{2i-1}) = 0 \bmod p$ and for NH $(x_{2i-1} + k_{2i-1}) = 0 \bmod 2^w$); in this case $g_k(x)$ is independent of x_{2i}. This attack requires p or 2^w text-MAC queries and a similar number of MAC verification queries to find a single key word k_i. With a divide-and-conquer approach recovering the complete key requires t times more work.

A slightly more efficient attack is described next. Assume that one knows an additive relation between two key words, say $k_{2i} = (k_{2i-1} + \Delta) \bmod 2^w$ for a $\Delta \in \mathbf{Z}_{2^w}$. Then if one replaces (x_{2i-1}, x_{2i}) by $((x_{2i} + \Delta) \bmod 2^w, (x_{2i-1} - \Delta) \bmod 2^w)$ the hash value will be unchanged. This implies that a single verification query allows to verify a guess for Δ. By trying all 2^w values of Δ, it follows that with 2^w verification queries one finds a relation between k_{2i} and k_{2i-1}. This relation is sufficient to perform subsequent forgeries. In order to find k_{2i} (and k_{2i-1}), one can apply the method of the previous paragraph; recovering the t key words requires only $t \cdot 2^w$ verification queries and $t \cdot 2^{w-2}$ MAC queries. Alternatively one can choose $x_{2i-1} = x_{2i} + \Delta$ (which makes the two factors equal) and apply the techniques of Square Hash with chosen texts.

If these functions are used with large word lengths (64 bits or more such as for VMAC), these observations do *not* pose a serious problem. However, NH has variants with $w = 32$ (for UMAC) and WH is being proposed with values of w equal to 16, 32 and 64. If one wants to obtain higher security levels, one can use two parallel instantiations and in this case the function will behave as if the word length was twice as large. However, if one wants to use single instantiations of the small word sizes for very high performance, a security level against forgery of $c \cdot 2^{-w}$ for a small constant c may be acceptable, but a partial key recovery attack with complexity on the order of 2^w clearly is not.

The above attacks shed a new light on the security of UMAC; even if the UMAC RFC [27] warns to restrict the number of MAC verifications, the document explicitly states that no key recovery attack is known that can exploit a forgery.

In order to conclude this section, we point out that the above attacks (except for the attack by Joux on GCM) apply to the three options and do not require nonce reuse.

3.3 Birthday collision attacks on polynomial hash functions

In this section we consider attacks on polynomial hash functions; these attacks require nonce reuse by the sender for Option 2 and 3 of Sect. 2.3; this implies that these attacks violate the standard security assumptions for these MAC algorithms. However, it is important to note that Option 1 has no nonce, hence it is obvious that for this case the attacks stay within the model.

If nonces are reused in polynomial hash schemes over GF(p), it is easy to recover the key k (Bernstein [6] is also very clear about the fact that in this case no security guarantees can be provided).

For polynomial hash over GF(2^n) and GCM, if we assume that somehow the sender can be convinced to reuse a nonce, or if we are using Option 1, we can enhance our previous attack by using the special messages of the Ferguson attack (we refer the reader to Annex A for a description of the original attack). As we no longer have access to the individual bits of $g_k(x) \oplus g_k(x')$ for two distinct messages x and x', we can only test equality between the complete n-bit strings $g_k(x)$ and $g_k(x')$. The birthday paradox allows to extend the attack.

1. consider messages with $x_i = 0$ except if $i = 2^j$ for some $j \geq 0$. As squaring in GF(2^n) is a linear operation in GF(2^n), the hash function becomes linear over this subset of inputs, hence we can write the bits of the hash result as follows:
$$g_k(x)[.] = \sum_{i*,j*,u*} x_{i*}[j*] \cdot k[u*] \,.$$
2. by guessing the linear combination of s ($1 \leq s < n$) well-chosen bits of the key k, we can generate a set of λ messages for which the hash result is

restricted to a subspace of size 2^{n-s} (e.g., the first s bits are equal to 0) (for details, see [16]);
3. now collect $\lambda = 2^{(n-s)/2+1}$ messages/MAC pairs (with the same nonce!) resulting in 2 collisions for the MAC value; each collision yields $n - s$ linear equations for the remaining $n - s$ key bits.

If we choose $n = 80$, $s = 24$, then $n - s = 56$: 2^{29} messages yield 2 collisions, resulting in 112 linear equations in the remaining 56 key bits. Note that even in the limit case of $s = 0$, this attack can recover the key k in $2^{n/2+1}$ chosen text/MAC pairs independently of k'.

4 Conclusion and Directions for Further Work

Table 1 presents an overview of the attacks. The security analysis of the many schemes and variants under multiple attacks is rather complex; we attempt here to summarize the main points. All universal hash functions except for the polynomial hash functions have large classes of weak keys, which have more serious implications than what was believed so far. All the universal hash functions that are based on key words that are substantially smaller than the output size (e.g, MMH, Square Hash and NMH with w-bit subkeys) allow – in spite of their large internal keys – for efficient divide-and-conquer key recovery attacks with approximately $t \cdot 2^w$ MAC verifications. While it is well understood that 2^w MAC verifications allow for a single forgery and – in the case of key reuse – for multiple forgeries [10], the implications of a key recovery attack are much more serious as they allow for arbitrary forgery later on. Most of the hash functions (except for polynomial hashing over GF(p)) allow for improved attacks if an oracle provides partial information on the secret key. It is important to stress that most of our attacks work for the three options (with and without nonces) and do not require nonce reuse: they are thus completely within the standard model for MAC algorithms. It is surprising that the more expensive Option 3 (called WMAC in [10]) does not offer an increased protection against key recovery attacks.

Overall, the polynomial hash functions over GF(p) seem to present fewer problems, but they are extremely vulnerable to nonce reuse, in particular if Option 2 (addition of a pseudo-random string) is used.

While universal hash functions have very attractive performance and provable security, our attacks demonstrate that most published MAC algorithms based on universal hash functions can be very brittle because of their simple combinatorial properties. Even within the security model, key recovery attacks are very efficient in spite of the large internal keys. Moreover, for those schemes for which no key recovery attack exists, a small violation of secure usage principles results in a collapse of security. This can be noted from very strong warnings in the specifications about nonce reuse and the detection of a large number of incorrect MAC verifications. A similar comment holds for partial key leakage (which can occur e.g., under side channel attacks). As it is very difficult to predict how a cryptographic algorithm will be used and abused, this is a highly undesirable property.

Table 1. Summary of our findings on MAC algorithms based on universal hash functions; if the key consists of small words, a word length of w is assumed. The third column indicates which part of k one needs to guess to allow for a verification with a single query; the fourth column indicates whether our attacks can exploit partial information on the key; the last column indicates the applicability of the new birthday attacks presented in this paper.

	number of weak keys	divide and conquer attacks	partial key information	new birthday attack
Polynomial hash GF(2^n)	1	k only	yes	yes
Polynomial hash GF(p)	1	k only	?	
Bucket hashing with small key	type I/II	subkey k_j	yes	yes
MMH	type I/II	w-bit subkey k_i	yes	
Square Hash	type II	w-bit subkey k_i	yes	
NMH/NH/WH	type II/III	w-bit subkey k_i	yes	

We present the following recommendations for MAC algorithms based on universal hash functions:

- Avoid reusing keys; while this may not be an option for schemes such as UMAC, for the polynomial hash functions, the overhead to generate a new value k for the universal hash function is quite small and brings a substantial increase of security and robustness as it will render most key recovery attacks useless; this approach is taken by SNOW 3G [1].
- In environments where side channel attacks are a concern, additional measures need to be taken which may negatively influence the performance.
- Sender and receiver need to guarantee/check uniqueness of nonces. The requirement for the sender seems to be more stringent in particular if a crash of the device needs to be considered. Appropriate implementation measures are necessary, such as storing counters in non-volatile memory. At the receiver side, restricting the number of MAC verifications with a single key seems more stringent than checking uniqueness. In any case, if random numbers are used to instantiate nonces, checking uniqueness for the receiver is problematic.

As an alternative for universal hash function based MACs, we recommend an AES-based scheme such as EMAC [18, 32]. It is somewhat slower but more "robust" in the following sense:

- Internal collisions (2^{64} texts) lead to forgeries, but not to key recovery.
- There is no known way to use an oracle that gives access to 32 or 64 key bits to speed-up key recovery by more than 2^{32} or 2^{64} operations.
- The algorithm benefits of a faster key setup.

We are currently investigating other universal hash functions, such as the new function proposed by Bernstein [7]. We also believe that it would be valuable to formalize the "regularity" of a universal hash function which could alleviate concerns related to weak keys.

Acknowledgements. The authors would like to thank the anonymous referees for helpful comments.

References

1. 3GPP TS 35.216, *"Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2; Document 2: SNOW 3G specification,"* March 2006.
2. M. Bellare, "New Proofs for NMAC and HMAC: Security without Collision-Resistance," *Proc. Crypto'06, LNCS 4117*, C. Dwork, Ed., Springer-Verlag, 2006, pp. 602–619.
3. M. Bellare, R. Canetti, H. Krawczyk, "Keying Hash Functions for Message Authentication," *Proc. Crypto'96, LNCS 1109*, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 1–15.
4. M. Bellare, O. Goldreich, A. Mityagin, "The Power of Verification Queries in Message Authentication and Authenticated Encryption," `http://eprint.iacr.org/2004/309`, Nov 18, 2004.
5. M. Bellare, J. Kilian, P. Rogaway, "The Security of Cipher Block Chaining," *Proc. Crypto'94, LNCS 839*, Y. Desmedt, Ed., Springer-Verlag, 1994, pp. 341–358.
6. D.J. Bernstein, "The Poly1305-AES message-authentication code," *Fast Software Encryption 2005, LNCS 3557*, H. Gilbert, H. Handschuh, Eds., Springer-Verlag, 2005, pp. 32–49. See slides at `http://cr.yp.to/talks/2005.02.15/slides.pdf`.
7. D.J. Bernstein, "Polynomial Evaluation and Message Authentication," `http://cr.yp.to/papers.html#pema`, 22 October 2007.
8. J. Bierbrauer, T. Johansson, G. Kabatianskii, B. Smeets, "On Families of Hash Functions via Geometric Codes and Concatenation," *Proc. Crypto'93, LNCS 773*, D. Stinson, Ed., Springer-Verlag, 1994, pp. 331–342.
9. J. Black, S. Halevi, H. Krawczyk, T. Krovetz, P. Rogaway. "UMAC: Fast and Secure Message Authentication," *Proc. Crypto'99, LNCS 1666*, M. Wiener, Ed., Springer-Verlag, 1999, pp. 216–233.
10. J. Black, M. Cochran "MAC Reforgeability," `http://eprint.iacr.org/2006/095`, 27 November 2007.
11. G. Brassard, "On Computationally Secure Authentication Tags Requiring Short Secret Shared Keys," *Proc. Crypto'82*, D. Chaum, R.L. Rivest, and A.T. Sherman, Eds., Plenum Press, New York, 1983, pp. 79–86.
12. J.L. Carter, M.N. Wegman, "Universal classes of hash functions," *Journal of Computer and System Sciences*, Vol. 18, 1979, pp. 143–154.
13. B. den Boer, "A Simple and Key-Economical Unconditional Authentication Scheme," *Journal of Computer Security*, Vol. 2, 1993, pp. 65–71.
14. Y. Dodis, K. Pietrzak, "Improving the Security of MACs via Randomized Message Preprocessing," *Fast Software Encryption, LNCS 4593*, A. Biryukov, Ed., Springer-Verlag, 2007, pp. 414–433.
15. M. Etzel, S. Patel, Z. Ramzan, "Square Hash: Fast Message Authentication via Optimized Universal Hash Functions," *Proc. Crypto'99, LNCS 1666*, M. Wiener, Ed., Springer-Verlag, 1999, pp. 234–251.
16. N. Ferguson, "Authentication Weaknesses in GCM," May 20, 2005. Available from `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/CWC-GCM/Ferguson2.pdf`.
17. S. Halevi, H. Krawczyk, "MMH: Software Message Authentication in the Gbit/second Rates," *Fast Software Encryption, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 172–189.

18. ISO/IEC 9797, *"Information Technology – Security Techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a Block Cipher,"* ISO/IEC 1999.
19. T. Iwata, K. Kurosawa, "OMAC: One-Key CBC MAC," *Fast Software Encryption, LNCS 2887*, T. Johansson, Ed., Springer-Verlag, 2003, pp. 129–153.
20. T. Johansson, "Bucket Hashing with a Small Key Size," *Proc. Eurocrypt'97, LNCS 1233*, W. Fumy, Ed., Springer-Verlag, 1997, pp. 149–162.
21. A. Joux, "Authentication Failures in NIST Version of GCM," 2006. Available from `http://csrc.nist.gov/CryptoToolkit/modes/`.
22. G.A. Kabatianskii, T. Johansson, B. Smeets, "On the Cardinality of Systematic A-codes via Error Correcting Codes," *IEEE Trans. on Information Theory*, Vol. IT–42, No. 2, 1996, pp. 566–578.
23. J.-P. Kaps, K. Yüksel, B. Sunar, "Energy Scalable Universal Hashing," *IEEE Trans. on Computers*, Vol. 54, No. 12, 2005, pp. 1484–1495.
24. L. Knudsen, "Chosen-text Attack on CBC-MAC," *Electronics Letters*, Vol. 33, No. 1, 1997, pp. 48–49.
25. T. Kohno, J. Viega, D. Whiting, "CWC: A High-Performance Conventional Authenticated Encryption Mode," *Fast Software Encryption 2004, LNCS 3017*, B.K. Roy, W. Meier, Eds., Springer-Verlag, 2004, pp. 408–426.
26. H. Krawczyk, "LFSR-based Hashing and Authentication," *Proc. Crypto'94, LNCS 839*, Y. Desmedt, Ed., Springer-Verlag, 1994, pp. 129–139.
27. T. Krovetz, "UMAC: Message Authentication Code using Universal Hashing," IETF, RFC 4418 (informational), March 2006.
28. T. Krovetz, "Message Authentication on 64-bit Architectures," *Selected Areas in Cryptography – SAC 2006, LNCS 4356*, E. Biham, A. Youssef, Eds., Springer-Verlag, 2007, pp. 327–341.
29. D.A. McGrew, S. Fluhrer, "Multiple Forgery Attacks against Message Authentication Codes," Available from `http://eprint.iacr.org/2005/161`.
30. D.A. McGrew, J. Viega, "The Security and Performance of the Galois/Counter Mode (GCM) of Operation," *Proc. Indocrypt 2004, LNCS 3348*, A. Canteaut, Ed., Springer-Verlag, 2004, pp. 343–355.
31. National Institute of Standards and Technology (NIST), SP 800-38D, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, November 2007 (earlier drafts published in May 2005, April 2006, June 2007).
32. E. Petrank, C. Rackoff, "CBC MAC for Real-time Data Sources," *Journal of Cryptology*, Vol. 13, No. 3, 2000, pp. 315–338.
33. B. Preneel, A. Bosselaers, R. Govaerts, J. Vandewalle, "A Chosen Text Attack on The Modified Cryptographic Checksum Algorithm of Cohen and Huang," *Proc. Crypto'89, LNCS 435*, G. Brassard, Ed., Springer-Verlag, 1989, pp. 154–163.
34. B. Preneel, P.C. van Oorschot, "On the Security of Iterated Message Authentication Codes," *IEEE Trans. on Information Theory*, Vol. IT–45, No. 1, 1999, pp. 188–199.
35. G.J. Simmons, "A Survey of Information Authentication," in *"Contemporary Cryptology: The Science of Information Integrity,"* G.J. Simmons, Ed., IEEE Press, 1991, pp. 381–419.
36. D.R. Stinson, "Universal Hashing and Authentication Codes," *Designs, Codes, and Cryptography*, Vol. 4, No. 4, 1994, pp. 369–380.
37. M.N. Wegman, J.L. Carter, "New Hash Functions and their Use in Authentication and Set Equality," *Journal of Computer and System Sciences*, Vol. 22, No. 3, 1981, pp. 265–279.

A The Joux and Ferguson Attacks on GCM

The Joux attacks on GCM. In his comment on the April 2006 draft of the NIST GCM mode [21], Joux presents a very elegant key recovery attack if the polynomial hash function is used with Option 2 under the condition of nonce reuse by the sender. The attack requires two text/MAC pairs with the same nonce and computes the exor of the tags, which is equal to $g_k(x) \oplus g_k(x')$ (the encryption with a one-time pad or pseudo-random function cancels out). This is a polynomial of degree at most t for which k is one of the roots. As there are at most t roots, it is easy to recover k, either by asking for a second set of text/MAC pairs (as proposed by Joux) or by using the technique of the first paragraph of Section 3.2; the latter approach requires at most t text/MAC pairs and t MAC verification queries (but without nonce reuse). Joux also shows that a small discrepancy in the specification between the May 2005 draft of [31] and the original GCM proposal of [31] leads to an efficient recovery of k *without nonce reuse*; this issue is solved in the June 2007 draft of [31]. In his comments, Joux suggests that *"replacing the counter encryption for MACs by the classical encryption with the block cipher usually used with Wegman-Carter MACs seems a safe option."* He also proposes a further mitigation of the security risks by using a stronger key derivation and by using separate keys for the different components of GCM.

The Ferguson attack on GCM. Ferguson points out another clever attack on GCM that exploits the arithmetic properties $\mathrm{GF}(2^n)$ with the truncation option. His attack consists of the following steps:

1. consider messages with $x_i = 0$ except if $i = 2^j$ for some $j \geq 0$. As squaring in $\mathrm{GF}(2^n)$ is a linear operation in $\mathrm{GF}(2^n)$, the hash function becomes linear over this subset of inputs (that is, the hash computation can be written as a matrix-vector product over GF(2));
2. for a single chosen text x of this form, modify the message into x' (again of the same form), where x' is chosen such that the first $s \leq \tau$ bits of $g_k(x)$ and $g_k(x')$ are equal, independent of the value of k. This can be achieved with some simple linear algebra with the constraint that $s < \log_2(t)$;
3. submit x' for verification: the probability of a successful forgery is $1/2^{\tau - s}$.

Moreover, a successful forgery results in $\tau - s$ additional linear equations in the bits of k, which makes it easier to obtain an additional forgery and thus even more information on the key.

Note that the GCM final standard [31] still uses Option 2 (in order to save a single encryption and avoid pipeline stalls) and still allows for truncation. On the other hand, it adds very explicit warnings about the risks of nonce reuse and truncation (in Appendices A and C respectively).