BTM: A Single-Key, Inverse-Cipher-Free Mode for Deterministic Authenticated Encryption

Tetsu Iwata[1] and Kan Yasuda[2]

[1] Dept. of Computational Science and Engineering, Nagoya University, Japan
iwata@cse.nagoya-u.ac.jp
[2] NTT Information Sharing Platform Laboratories, NTT Corporation, Japan
yasuda.kan@lab.ntt.co.jp

Abstract. We present a new blockcipher mode of operation named BTM, which stands for Bivariate Tag Mixing. BTM falls into the category of Deterministic Authenticated Encryption, which we call DAE for short. BTM makes all-around improvements over the previous two DAE constructions, SIV (Eurocrypt 2006) and HBS (FSE 2009). Specifically, our BTM requires just one blockcipher key, whereas SIV requires two. Our BTM does not require the decryption algorithm of the underlying blockcipher, whereas HBS does. The BTM mode utilizes *bivariate* polynomial hashing for authentication, which enables us to handle vectorial inputs of dynamic dimensions. BTM then generates an initial value for its counter mode of encryption by *mixing* the resulting *tag* with one of the two variables (hash keys), which avoids the need for an implementation of the inverse cipher.

Keywords: Bivariate, universal hash function, counter mode, random-until-bad game, systematic proof.

1 Introduction

The modes of operation for blockciphers can be divided into the following three groups: encryption modes, message authentication codes, and authenticated encryption modes. The first group achieves privacy, the second ensures integrity, and the third does both at the same time. On one hand, the third group is attractive to users for its providing both kinds of security concurrently. On the other hand, the third group tends to employ rather complex mechanisms, since authenticated encryption modes are essentially a combination of an encryption mode and a message authentication code [2].

The complexity of authenticated encryption adds variety to the constructions, such as CCM [16], GCM [10] and OCB [14]. These constructions, however, have one thing in common—their security is based on the so-called nonce assumption. The assumption requires that nonce values be never repeated. Otherwise, the security of the overlying scheme is seriously compromised, which is a difficult aspect of the nonce-based constructions.

The problem of the nonce assumption was settled by the notion of Deterministic Authenticated Encryption (DAE), which Rogaway and Shrimpton introduced

M.J. Jacobson Jr., V. Rijmen, and R. Safavi-Naini (Eds.): SAC 2009, LNCS 5867, pp. 313–330, 2009.

at Eurocrypt 2006 [15]. DAE constructions are more robust than nonce-based ones. Namely, a DAE construction can be used as a nonce-based one by embedding a nonce value into part of its input, in which case the DAE construction achieves the same security level as a nonce-based one. Furthermore, a DAE construction maintains a certain level of security even when no nonce value is combined with. In such a scenario, an adversary can detect a repetition of exactly the same inputs being encrypted, but nothing more.

Though more robust, DAE modes are also more difficult to construct than nonce-based ones. To date, there have been two concrete DAE constructions, SIV [15] and HBS [7]. The SIV construction utilized blockcipher iteration both for its encryption algorithm and for authentication. SIV had a number of attractive features but had one major disadvantage that it required two blockcipher keys. This disadvantage was removed by the more recent single-key HBS construction. HBS also accelerated the speed by employing polynomial hashing, rather than blockcipher iteration, for its authentication algorithm.

Nevertheless, at the same time, the HBS construction sacrificed many of SIV's advantages in the interest of single-key usage and of polynomial-hashing design. In the following we point out those disadvantages which HBS suffered.

1. Inverse-Cipher Requirement. The decryption process of the HBS mode required the decryption algorithm of the underlying blockcipher. This requirement involved numerous drawbacks. First, HBS increased the size of its footprint (e.g., the number of gates or slices). Second, the security proof of HBS relied on the stronger SPRP (Strong Pseudo-Random Permutation) assumption about the underlying blockcipher. Third, the tag size of HBS was fixed to the full n-bit, disabling any kind of tag truncation for saving the bandwidth. These problems did not exist within the SIV construction, which worked without the inverse cipher.
2. Worse Security Bound. The security bound of HBS was of the form $\ell^2 q^2/2^n$, where ℓ denotes the maximum length of each query and q the total number of queries. This should be contrasted with the security bound of SIV which was of the form $\sigma^2/2^n$, where σ denotes the maximum query complexity (i.e., the total length of all queries). The former becomes markedly worse than the latter when queries are of varying lengths.
3. Inflexible Vector Dimensions. HBS needed to fix in advance the dimension of vectorial inputs (i.e., the maximum number of headers). Note that handling flexible vector dimensions is an important advantage, because some applications may be unable to set a limit reflecting the unpredictable nature of the dimension, or some applications may want to update and increase the limit on the dimension while maintaining backward compatibility. HBS suggested using the square-root operation $\sqrt{\ }$ as a remedy for dynamic changes in the dimension, but such a technique was more complex and less elegant than the vectorized-CMAC [6,13] solution offered by SIV.
4. Enlarged Counter Register. The HBS mode specified an unusual increment method $S \oplus \underline{1}_n$, $S \oplus \underline{2}_n$, ..., where S denotes an initial value, $\oplus$ the bitwise xor operation, and $\underline{a}_n$ an n-bit binary representation of an integer a.

Table 1. Comparison between SIV, HBS and BTM. The figures show the number of computations for a single header H and a message M, where $h = \lceil |H|/n \rceil$ and $m = \lceil |M|/n \rceil$.

	SIV	HBS	BTM
# of blockcipher keys	2	1	1
Inverse-cipher-free	yes	no	yes
Blockcipher assumption	PRP	SPRP	PRP
Tag truncation	possible	impossible	possible
Security bound	$O(\sigma^2/2^n)$	$O(\ell^2 q^2/2^n)$	$O(\sigma^2/2^n)$
Vector dimension	dynamic	static	dynamic
Counter register size	n-bit	$1.5n$-bit	n-bit
Total # of computations	$h + 2m + 2$	$h + 2m + 4$	$h + 2m + 2$
# of blockcipher calls	$h + 2m + 2$	$m + 2$	$m + 3$
# of multiplications	0	$h + m + 2$	$h + m - 1$

This method required a $1.5n$-bit register in order to maintain the current counter value, namely n bits for keeping the value S and $0.5n$ bits for incrementing a. On the other hand, SIV worked with almost any increment method, such as $S+1$, $S+2$, ... (arithmetic addition modulo 2^n) and $\underline{2}_n \cdot S$, $\underline{2}_n^2 \cdot S$, ... (doubling in the finite field of 2^n elements). These usual methods require only an n-bit register and maintain smaller sizes of footprints.

5. Increased Number of Computations. Although HBS accelerated the speed for large data by employing polynomial hashing, the total number of computations, which is the sum of the number of blockcipher calls plus that of finite-field multiplications, increased by 2. This caused the degradation in performance for short messages, depending on implementations.

The goal of the current work is to overstep this line of tradeoff and to construct a new single-key, polynomial-hashing DAE mode of operation which does not counterbalance any of the advantages that SIV had. For this, we propose a new mode of operation called BTM, which stands for Bivariate Tag Mixing. See Table 1 for summary. The BTM construction achieves our goal by utilizing the following two techniques.

1. Bivariate Polynomial Hashing. Our polynomial hashing utilizes two variables (keys) L and U, which are derived from a single blockcipher key K as $L := E_K(\underline{0}_n)$ and $U := E_K(\underline{1}_n)$. The bivariate hashing is capable of handling vectorial inputs of dynamic dimensions. The bivariate hashing also contributes to reducing the number of finite-field multiplications.
2. Tag Mixing. The tag value T is mixed with the hash key U via a special type of arithmetic addition, for which we write $T \boxplus U$. The mixed value is used as an initial value for the counter mode of encryption. In this way we avoid the use of the inverse cipher. In addition, this mixing allows us to use the increment $\boxplus \underline{1}_n$, saving the register size of the counter.

2 Preliminaries

Notation. We have already introduced the symbols $X \oplus Y$, $\underline{a}_n$ and $X \cdot Y$. The symbol $X \| Y$ denotes the concatenation of two strings X and Y. Given a string X, we write $|X|$ to represent its length in bits. If X is a finite set, then $|X|$ denotes its cardinality. We write $x \xleftarrow{\$} X$ for sampling an element from the set X uniformly at random and assigning its value to the variable x. Given a positive integer a and a string X such that $a \leq |X|$, $\mathrm{msb}(a, X)$ represents the leftmost a bits of the string X. The set $\{0,1\}^n$ of n-bit strings is regarded in multiple ways. It corresponds to the set $\{0, 1, \ldots, 2^n - 1\}$ of non-negative integers less than 2^n. It is also treated as the finite field $GF(2^n)$ of 2^n elements (with respect to some irreducible polynomial).

Blocks and Vectors. Throughout the paper we fix the *block size* n. Typical values of n are 64 and 128. The *block decomposition* $X[0] \cdots X[x-1]$ of a string $X \in \{0,1\}^*$ is computed as follows. If $X = \emptyset$ (the null string), then we set $x \leftarrow 1$ and $X[0] \leftarrow \emptyset$. Otherwise (i.e., when $|X| \geq 1$), we set $x \leftarrow \lceil |X|/n \rceil$, and blocks $X[0], \ldots, X[x-1]$ are defined as the unique set of strings satisfying the conditions $X[0] \,\|\, \cdots \,\|\, X[x-1] = X$, $\big|X[0]\big| = \cdots = \big|X[x-2]\big| = n$, and $1 \leq \big|X[x-1]\big| \leq n$. We call x the *block length* of the string X.

Given a string $X \in \{0,1\}^*$ such that $|X| \leq n$, we define

$$\pi(X) := \begin{cases} X \| 1 \| 0^{n-1-|X|} & \text{if } 0 \leq |X| \leq n-1, \\ X & \text{if } |X| = n, \end{cases}$$

so that we always have $\big|\pi(X)\big| = n$. Similarly, we define

$$\delta(X) := \begin{cases} \underline{1}_n & \text{if } 0 \leq |X| \leq n-1, \\ \underline{2}_n & \text{if } |X| = n. \end{cases}$$

We consider d-dimensional vectors $\overrightarrow{X} = (X_0, \ldots, X_{d-2}, X_{d-1})$ of strings $X_i \in \{0,1\}^*$ for $i = 0, \ldots, d-2, d-1$.

Headers and Messages. A plaintext (i.e., data which is input to the encryption algorithm) consists of header information $\overrightarrow{H} = (H_0, \ldots, H_{d-2})$ and a message M, being a d-dimensional vector $\big(\overrightarrow{H}, M\big) = (H_0, \ldots, H_{d-2}, M)$ of strings. In the paper it is understood that when $d = 1$ the notation $\big(\overrightarrow{H}, M\big)$ represents the 1-dimensional vector (M) (i.e., the case of no header information). Note that the message part M of a plaintext gets both authenticated and encrypted, while header information $\overrightarrow{H}$ gets authenticated but remains unencrypted.

Blockciphers and DAEs. A blockcipher is a family of permutations. We often write $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ for a blockcipher, where $K \in \{0,1\}^k$ is a key and $E_K := E(K, \cdot) : \{0,1\}^n \to \{0,1\}^n$ is the permutation specified

by the key K. An adversary A is an oracle machine that outputs a bit. The goal of a PRP(Pseudo-Random Permutation)-adversary A is to distinguish the blockcipher E_K (with a random key K) from a truly random permutation $P : \{0,1\}^n \to \{0,1\}^n$ [9,1]. The success probability of A is measured by

$$\mathbf{Adv}^{\mathrm{prp}}_{E}(A) := \Pr\big[A^{E_K(\cdot)} = 1\big] - \Pr\big[A^{P(\cdot)} = 1\big],$$

where in the first game A has access to the E_K oracle and in the second the P oracle. We fix a model of computation and a choice of encoding. We write $\mathbf{Adv}^{\mathrm{prp}}_{E}(t,\sigma) := \max_A \mathbf{Adv}^{\mathrm{prp}}_{E}(A)$, where max runs over adversaries A whose time complexity is at most t and whose query complexity is at most σ. The time complexity is the running time plus the code size. The query complexity is the total length in blocks of the queries made to the oracles.

A DAE scheme is a pair of algorithms $\mathcal{E}_K$ and $\mathcal{D}_K$. The encryption algorithm $\mathcal{E}_K$ takes a vectorial input $\big(\overrightarrow{H}, M\big)$ and outputs a pair of a tag T and a ciphertext C, where $|T| = n$ and $|C| = |M|$. The decryption $\mathcal{D}_K$ takes an input $\big(\overrightarrow{H}, T, C\big)$ and outputs either the corresponding plaintext M or a special symbol $\perp$. The goal of a DAE-adversary A is to distinguish between the pair $(\mathcal{E}_K, \mathcal{D}_K)$ and the pair $(\mathcal{R}, \perp)$, where $\mathcal{R}$ is an oracle that returns, upon a query $\big(\overrightarrow{H}, M\big)$, random strings of $n + |M|$ bits, and $\perp$ is an oracle that always returns the $\perp$ symbol. We define

$$\mathbf{Adv}^{\mathrm{dae}}_{\mathcal{E},\mathcal{D}}(A) := \Pr\big[A^{\mathcal{E}_K(\cdots),\mathcal{D}_K(\cdots)} = 1\big] - \Pr\big[A^{\mathcal{R}(\cdots),\perp(\cdots)} = 1\big],$$

where trivial queries are excluded. As before we also define $\mathbf{Adv}^{\mathrm{dae}}_{\mathcal{E},\mathcal{D}}(t,\sigma)$.

3 Bivariate Polynomials and *L*-Polynomials

A *bivariate polynomial* is a polynomial in two variables L and U over the field of 2^n elements, i.e., an element of $GF(2^n)[L,U]$. A function G of two arguments L and U is said to be an *L-polynomial* if the following conditions are satisfied.

1. $G(L,U)$ is a polynomial in the variable L. Let x be the degree of $G(L,U)$ as a polynomial in L.
2. We then have $x \geq 1$.
3. The coefficient of the leading term L^x is a polynomial function of U. Let y be the degree of this coefficient function as a polynomial in U.

We define $\deg_L G := x$ and $\deg^U_L G := y$. Observe that any non-constant bivariate polynomial is either an L-polynomial or a U-polynomial (or both). Also note that if G is a bivariate polynomial, then we have $\deg^U_L G \leq \deg_U G$. Now the following lemma is a basic result, and its proof can be found in Appendix A.

Lemma 1. *Let G be an L-polynomial. We have*

$$\Pr\big[G(L,U) = \underline{0}_n \;\big|\; L \xleftarrow{\$} \{0,1\}^n, U \xleftarrow{\$} \{0,1\}^n\big] \leq \frac{\deg^U_L G + \deg_L G}{2^n}.$$

4 Specification of BTM

In this section we give the specification of the BTM algorithms. First, we define the bivariate polynomial hashing $F_{L,U}$. Second, we describe the way of mixing the tag value with one of the hash keys. Finally, we give the definition of the BTM encryption and decryption algorithms.

4.1 Bivariate Hashing $F_{L,U}$

We begin with defining a polynomial f_L in one variable L. Given a string $X \in \{0,1\}^*$, define

$$f_L(X) := \delta\big(X[x-1]\big) \cdot \big(L^x \oplus L^{x-1} \cdot X[0] \oplus \cdots \oplus L \cdot X[x-2] \oplus \pi(X[x-1])\big),$$

where the addition and the multiplication are done in the finite field $GF(2^n)$ of 2^n elements, so that $f_L(X)$ is an element of $GF(2^n)[L]$. Recall that the polynomial $f_L(X)$ can be computed recursively, using the relation

$$f_L(X) = \delta\big(X[x-1]\big) \cdot \big(\big(\cdots\big((L \oplus X[0]) \cdot L \oplus X[1]\big)\cdots\big) \cdot L \oplus \pi(X[x-1])\big).$$

Note that the polynomial $f_L(X)$ always has a degree x due to the leading term L^x.

Now we define the polynomial $F_{L,U}$ in two variables L and U as

$$F_{L,U}\big(\overrightarrow{H}, M\big) := U^{d-1} \cdot f_L(H_0) \oplus \cdots \oplus U \cdot f_L(H_{d-2}) \oplus f_L(M),$$

which is an element of $GF(2^n)[L,U]$. Roughly speaking, we first hash each of $H_0, H_1, \ldots, H_{d-2}, M$ in terms of the variable L, which results in d-many hash values $f_L(H_0)$, ..., $f_L(M)$, and then we hash these values in terms of the variable U. See Fig. 1 for an illustration.

Observe that $f_L(X)$ requires $x-1$ multiplications by L. Neglecting the xor operations and the last multiplication by $\underline{1}_n/\underline{2}_n$, the computation of $f_L(X)$ can

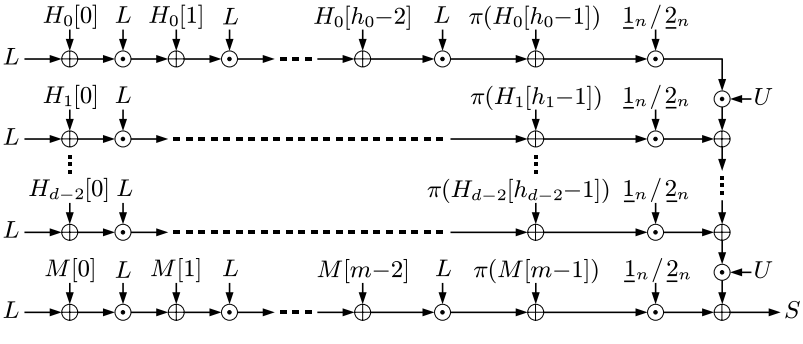

Fig. 1. Illustration of the bivariate hashing $F_{L,U}(H_0, H_1, \ldots, H_{d-2}, M)$. The symbol $X \odot Y$ denotes the multiplication $X \cdot Y$ in the field of 2^n elements. The multiplication by $\underline{1}_n/\underline{2}_n$ corresponds to the function $\delta\big(\pi(X[x-1])\big)$.

Algorithm $\mathrm{BTM.Enc}_K(\overrightarrow{H}, M)$

1. $L \leftarrow E_K(\underline{0}_n)$, $U \leftarrow E_K(\underline{1}_n)$
2. $T \leftarrow \mathrm{MAC}_K^{L,U}(\overrightarrow{H}, M)$
3. $C \leftarrow \mathrm{CTR}_K^{T \boxplus U}(M)$
4. **return** (T, C)

Algorithm $\mathrm{BTM.Dec}_K(\overrightarrow{H}, (T, C))$

1. $L \leftarrow E_K(\underline{0}_n)$, $U \leftarrow E_K(\underline{1}_n)$
2. $M \leftarrow \mathrm{CTR}_K^{T \boxplus U}(C)$
3. $T' \leftarrow \mathrm{MAC}_K^{L,U}(\overrightarrow{H}, M)$
4. **if** $T \neq T'$ **then**
5. $\quad M \leftarrow \perp$
6. **end if**
7. **return** M

Subroutine $\mathrm{MAC}_K^{L,U}(\overrightarrow{H}, M)$

1. $S \leftarrow F_{L,U}(\overrightarrow{H}, M)$
2. $T \leftarrow E_K(S)$
3. **return** T

Subroutine $\mathrm{CTR}_K^N(X)$

1. $x \leftarrow \lceil |X|/n \rceil$
2. **for** $i \leftarrow 0$ **to** $x - 1$ **do**
3. $\quad R[i] \leftarrow E_K(N \boxplus \underline{i}_n)$
4. **end for**
5. $R \leftarrow R[0] \,\|\, \cdots \,\|\, R[x-1]$
6. $Y \leftarrow X \oplus \mathrm{msb}(|X|, R)$
7. **return** Y

Fig. 2. Pseudocode of the BTM encryption and decryption algorithms. The subroutines MAC and CTR are extracted from the algorithms and shown on the right-hand side.

be done in about $x - 1$ finite-field multiplications. This means that for a two-dimensional vector (H, M) the computation of $F_{L,U}(H, M)$ can be done in about $(h-1)+(m-1)+1 = h+m-1$ finite-field multiplications (The last "+1" comes from the multiplication by U). This explains the figures in Table 1. More generally, for a d-dimensional vector $(\overrightarrow{H}, M)$, it takes about

$$(h_0-1)+(h_1-1)+\cdots+(h_{d-2}-1)+(m-1)+d-1 = h_0+h_1+\cdots+h_{d-2}+m-1$$

finite-field multiplications to compute the bivariate hashing $F_{L,U}(\overrightarrow{H}, M)$ (the value h_i being the block length of H_i and m that of M).

It can be directly verified that if $(\overrightarrow{H}, M) \neq (\overrightarrow{H}', M')$ are two distinct inputs, then we have an inequality of polynomials $F_{L,U}(\overrightarrow{H}, M) \neq F_{L,U}(\overrightarrow{H}', M')$. This fact plays an important role in our security analysis.

4.2 Tag Mixing $T \boxplus U$

The operation $\boxplus$ is defined as follows. For two strings $X, Y \in \{0,1\}^n$, divide them into two equal-length parts as $X = X_1 \| X_2$ and $Y = Y_1 \| Y_2$, so that $X_1, X_2, Y_1, Y_2 \in \{0,1\}^{n/2}$.[1] Then define

$$X \boxplus Y := (X_1 + Y_1) \| (X_2 + Y_2),$$

where the addition $+$ is done modulo $2^{n/2}$. We use $\boxplus$ rather than $+$ modulo 2^n, because $\boxplus$ is less costly and is sufficient for security up to the birthday bound.

[1] We assume that the block size n is an even number.

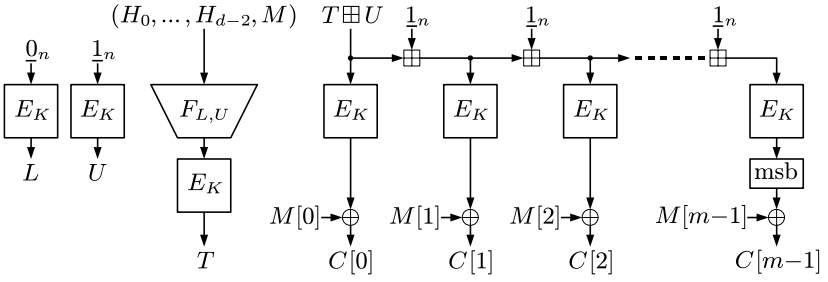

Fig. 3. Illustration of the BTM encryption algorithm

4.3 BTM Encryption and Decryption Algorithms

We are now ready to describe our BTM mode of operation. The encryption algorithm BTM.Enc and the decryption BTM.Dec are described in Fig. 2. See also Fig. 3 for a diagrammatic representation of the BTM encryption algorithm.

Note that when $|M| = 0$ (i.e., $M = \emptyset$ the null string), the algorithm BTM.Enc returns only the tag T. Also note that the one-dimensional input (M) (without a header) and the two-dimensional input (H, M) with $H = \emptyset$ generally result in different outputs for the same value of M.

5 Security Analysis of BTM

We prove the security of our BTM mode as a DAE construction. We first introduce a simple tool which makes our analysis easy. We then prove the privacy and the integrity of BTM. Thanks to the tool, our proofs are quite systematic, consisting of counting "bad" events and computing their probabilities.

5.1 A Simple Tool: Random-Until-Bad Games

We consider a special type of game called a "random-until-**bad**" game. This type of game can be systematically analyzed, making our security proofs simple and easy.

In a random-until-bad game, the adversary's goal is to set a **bad** flag written somewhere in the description of the overlying game. There may be multiple **bad** flags, as **bad**[0], **bad**[1], etc. The adversary wins the game as soon as one of the **bad** flags gets set. The only way for the adversary to set a **bad** flag is by making a query to its oracle. A query is processed according to the description of the game. If a query sets a **bad** flag, then the game terminates immediately. Otherwise, the oracle returns a truly random string of a specified length[2] to the adversary.

In a random-until-bad game, we only need to consider non-adaptive adversaries. Recall that oracles only return random strings to an adversary until the

[2] We consider the special symbol $\perp$ as a random string of length zero.

game terminates. This means that any adaptive adversary can be transformed into a non-adaptive one without changing its winning probability, by feeding a random tape to the adversary. More precisely, let A be an adaptive adversary which makes (exactly) q queries. Using A, we can construct a non-adaptive adversary B which has about the same running time as A and exactly the same winning probability, as follows: We let B run A by simulating A's oracles via B's internal random coins. The adversary B records A's queries $x_1, x_2, \ldots, x_q$. Then B outputs the sequence of queries $x_1, x_2, \ldots, x_q$. Note that at this point B has made no queries to B's oracles, and the values of the queries $x_1, x_2, \ldots, x_q$ have been already fixed. Hence we see that B is non-adaptive. It is also easy to see that B's winning probability is exactly the same as that of A.

Furthermore, we only need to consider deterministic adversaries. For this, we show that for any non-adaptive, probabilistic adversary there exists a non-adaptive, deterministic adversary having the same or better winning probability. So let A be a non-adaptive, probabilistic adversary making q queries to its oracles. For each sequence of queries $x_1, x_2, \ldots, x_q$ (the total length being no more than σ) the adversary A outputs this sequence with some probability. The winning probability of A is the weighted average (arithmetic mean) of the winning probabilities over all sequences $x_1, x_2, \ldots, x_q$. Then there exists a sequence $x_1^*, x_2^*, \ldots, x_q^*$ having the maximum winning probability. So let A^* be the adversary that always outputs the sequence $x_1^*, x_2^*, \ldots, x_q^*$. Then we see that the winning probability of A^* is no less than that of A.

In a random-until-bad game, we can systematically compute each probability that a **bad** flag gets set and then sum up the probabilities. This gives us the bound of the adversaries' winning probability.

5.2 From Computational to Information-Theoretic

The first step is to replace the blockcipher E_K (using a random key K) with a random permutation $P : \{0,1\}^n \to \{0,1\}^n$. We write BTM[$P$] for such a DAE scheme. Let A be a DAE-adversary whose time and query complexities are at most t and σ, respectively. We can directly construct an adversary B that uses A and tries to distinguish between the blockcipher E_K and the random permutation P. The simulation requires two calls to E for computing L and U, q calls to E for computing tags (together with necessary polynomial hashing), and σ calls to E for encrypting the messages. Any difference between $\mathbf{Adv}^{\mathrm{dae}}_{\mathrm{BTM}[E]}(A)$ and $\mathbf{Adv}^{\mathrm{dae}}_{\mathrm{BTM}[P]}(A)$ contributes to B's advantage $\mathbf{Adv}^{\mathrm{prp}}_{E}(B)$, so we have

$$\mathbf{Adv}^{\mathrm{dae}}_{\mathrm{BTM}[E]}(t,\sigma) \leq \mathbf{Adv}^{\mathrm{prp}}_{E}(t', 2+q+\sigma) + \mathbf{Adv}^{\mathrm{dae}}_{\mathrm{BTM}[P]}(\sigma),$$

where the running time t' is about t plus the complexity to compute $2+q+\sigma$ times the blockcipher E (and the complexity to perform corresponding polynomial hashing). Note that we have omitted the time complexity from the notation, since it becomes irrelevant to the context of BTM[P].

We then replace P with a random *function* $F : \{0,1\}^n \to \{0,1\}^n$ (not to be confused with $F_{L,U}$). Using the PRP/PRF switching lemma [3], we obtain

$$\mathbf{Adv}^{\mathrm{dae}}_{\mathrm{BTM}[P]}(\sigma) \le \binom{2+q+\sigma}{2} \cdot \frac{1}{2^n} + \mathbf{Adv}^{\mathrm{dae}}_{\mathrm{BTM}[F]}(\sigma).$$

Therefore, it amounts to evaluating the security of the scheme BTM$[F]$.

Now let $\mathcal{E}_F$ and $\mathcal{D}_F$ denote the encryption and decryption algorithms of the BTM$[F]$ scheme, respectively. For an adversary A we have

$$\begin{aligned}\mathbf{Adv}^{\mathrm{dae}}_{\mathrm{BTM}[F]}(A) &= \Pr\left[A^{\mathcal{E}_F(\cdots),\mathcal{D}_F(\cdots)} = 1\right] - \Pr\left[A^{\mathcal{R}(\cdots),\perp(\cdots)} = 1\right] \\ &= \Pr\left[A^{\mathcal{E}_F(\cdots),\mathcal{D}_F(\cdots)} = 1\right] - \Pr\left[A^{\mathcal{E}_F(\cdots),\perp(\cdots)} = 1\right] \quad \text{(integrity)} \\ &\quad + \Pr\left[A^{\mathcal{E}_F(\cdots),\perp(\cdots)} = 1\right] - \Pr\left[A^{\mathcal{R}(\cdots),\perp(\cdots)} = 1\right]. \quad \text{(privacy)}\end{aligned}$$

We shall evaluate the privacy first and then integrity.

5.3 Privacy Proof of BTM[F]

Theorem 1. *Let A be an adversary whose total query complexity is at most σ blocks. Then we have*

$$\Pr\left[A^{\mathcal{E}_F(\cdots),\perp(\cdots)} = 1\right] - \Pr\left[A^{\mathcal{R}(\cdots),\perp(\cdots)} = 1\right] \le \frac{8\sigma^2}{2^n}.$$

Proof. Consider the eight **bad** events listed in Table 2. These **bad** flags are placed in the description of the $\mathcal{E}_F$ oracle. Observe that these **bad** events cause the $\mathcal{E}_F$ oracle to return some non-random values by invoking the function F on some "old" inputs. In other words, as long as none of these **bad** flags gets set, the $\mathcal{E}_F$ oracle behaves exactly the same as the ideal $\mathcal{R}$ oracle, since the values returned by the $\mathcal{E}_F$ oracle are then outputs of the random function F on some fresh inputs.

Therefore, by the fundamental lemma of game playing [3], we get

$$\Pr\left[A^{\mathcal{E}_F(\cdots),\perp(\cdots)} = 1\right] - \Pr\left[A^{\mathcal{R}(\cdots),\perp(\cdots)} = 1\right] \le \Pr\left[A^{\mathcal{E}_F(\cdots),\perp(\cdots)} \text{ sets } \mathbf{bad}\right],$$

and now the game under consideration is random-until-**bad**. Hence, we can systematically compute the winning probability of A. We simply sum up the probabilities in Table 2 as

$$\Pr\left[A^{\mathcal{E}_F(\cdots),\perp(\cdots)} \text{ sets } \mathbf{bad}\right] \le \frac{\sigma + \cdots + 2(q-1)(\sigma - q)}{2^n} \le \frac{8\sigma^2}{2^n},$$

which gives us the desired bound. □

5.4 Integrity Proof of BTM[F]

Theorem 2. *Let A be an adversary whose total query complexity is at most σ blocks. Then we have*

$$\Pr\left[A^{\mathcal{E}_F(\cdots),\mathcal{D}_F(\cdots)} = 1\right] - \Pr\left[A^{\mathcal{E}_F(\cdots),\perp(\cdots)} = 1\right] \le \frac{11\sigma^2}{2^n}.$$

Table 2. The **bad** events in the privacy game. The superscript (i) means that the variable comes from the i-th query. The adversary makes q queries. The indices run over $1 \le i \le q$, $1 \le j \le i-1$, $0 \le \alpha \le m^{(i)}-1$ and $0 \le \beta \le m^{(j)}-1$, where $m^{(i)}$ is the length in blocks of the queried message $M^{(i)}$. The computations of the probabilities can be found in Appendix B.

flag	event	type	probability
bad[0]	$S^{(i)} = \underline{0}_n$		$\sigma/2^n$
bad[1]	$S^{(i)} = \underline{1}_n$		$\sigma/2^n$
bad[2]	$S^{(i)} = S^{(j)}$	bivariate polynomial	$(q-1)\sigma/2^n$
bad[3]	$S^{(i)} = T^{(j)} \boxplus U \boxplus \underline{\beta}_n$	L-polynomial	$\sigma^2/2^n$
bad[4]	$T^{(i)} \boxplus U \boxplus \underline{\alpha}_n = \underline{0}_n$		$\sigma/2^n$
bad[5]	$T^{(i)} \boxplus U \boxplus \underline{\alpha}_n = \underline{1}_n$		$\sigma/2^n$
bad[6]	$T^{(i)} \boxplus U \boxplus \underline{\alpha}_n = S^{(j)}$	L-polynomial	$\sigma^2/2^n$
bad[7]	$T^{(i)} \boxplus U \boxplus \underline{\alpha}_n = T^{(j)} \boxplus U \boxplus \underline{\beta}_n$		$2(q-1)(\sigma-q)/2^n$

Proof. The two games are identical unless the $\mathcal{D}_F$ oracle returns something other than $\perp$. Therefore, by the fundamental lemma of game playing, we have

$$\Pr\left[A^{\mathcal{E}_F(\cdots),\mathcal{D}_F(\cdots)} = 1\right] - \Pr\left[A^{\mathcal{E}_F(\cdots),\perp(\cdots)} = 1\right] \le \Pr\left[A^{\mathcal{E}_F(\cdots),\mathcal{D}_F(\cdots)} \text{ forges}\right].$$

Using the adversary A, we shall construct a new adversary B that finds a forgery of the message authentication code $\mathcal{G}_F : \left(\overrightarrow{H}, M\right) \mapsto T$. We let B gain access to an auxiliary oracle $\mathcal{O}_F$, which returns the value $F(U \boxplus W)$ upon a query $W \in \{0,1\}^n$. Here recall that $U := F(\underline{0}_n)$. We let B simulate oracles for A in the natural way. This yields

$$\Pr\left[A^{\mathcal{E}_F(\cdots),\mathcal{D}_F(\cdots)} \text{ forges}\right] \le \Pr\left[B^{\mathcal{G}_F(\cdots),\mathcal{V}_F(\cdots),\mathcal{O}_F(\cdot)} \text{ forges}\right],$$

where $\mathcal{V}_F$ is the verification oracle of the message authentication code $\mathcal{G}_F$. We note that B makes at most $\sigma - q$ queries to the $\mathcal{O}_F$ oracle.

Now we introduce a random oracle $\mathcal{R}_n^n : \{0,1\}^n \to \{0,1\}^n$ (independent of F) and replace $\mathcal{O}_F$ with the ideal $\mathcal{R}_n^n$. Then we have

$$\begin{aligned}\Pr&\left[B^{\mathcal{G}_F(\cdots),\mathcal{V}_F(\cdots),\mathcal{O}_F(\cdot)} \text{ forges}\right] \\ &\le \Pr\left[B^{\mathcal{G}_F(\cdots),\mathcal{V}_F(\cdots),\mathcal{O}_F(\cdot)} \text{ forges}\right] - \Pr\left[B^{\mathcal{G}_F(\cdots),\mathcal{V}_F(\cdots),\mathcal{R}_n^n(\cdot)} \text{ forges}\right] \quad (1) \\ &\qquad + \Pr\left[\tilde{B}^{\mathcal{G}_F(\cdots),\mathcal{V}_F(\cdots)} \text{ forges}\right], \quad (2)\end{aligned}$$

where $\tilde{B}$ is the adversary that runs B by simulating the $\mathcal{R}_n^n$ oracle using $\tilde{B}$'s internal random coins.

First we evaluate the quantity (1). The two games proceed exactly the same as long as the oracle $\mathcal{O}_F$ returns only random strings. Consider the **bad** flags listed in Table 3. The flags **bad**$[8-11]$ are placed in the description of the $\mathcal{O}_F$ oracle, with the hash values $S^{(i)}$ being recorded upon queries to the $\mathcal{G}_F$ and $\mathcal{V}_F$

Table 3. The **bad** events in the integrity game. The index runs over $1 \le r \le \sigma - q$. The computations of the probabilities can be found in Appendix C.

flag	event	type	probability
bad[8]	$W^{(r)} \boxplus U = \underline{0}_n$		$(\sigma - q)/2^n$
bad[9]	$W^{(r)} \boxplus U = \underline{1}_n$		$(\sigma - q)/2^n$
bad[10]	$W^{(r)} \boxplus U = S^{(j)}$	L-polynomial	$(\sigma - q)(\sigma + q)/2^n$
bad[11]	$S^{(i)} = W^{(r)} \boxplus U$	L-polynomial	$(\sigma - q)(\sigma + q)/2^n$
bad[12]	$S^{(i)} = \underline{0}_n$		$\sigma/2^n$
bad[13]	$S^{(i)} = \underline{1}_n$		$\sigma/2^n$
bad[14]	$S^{(i)} = S^{(j)}$	bivariate polynomial	$(q-1)\sigma/2^n$
bad[15]	$\mathcal{V}_F(\cdots) \neq \perp$		$q/2^n$

oracles. The $\mathcal{O}_F$ oracle behaves just like the ideal $\mathcal{R}_n^n$ unless one of these **bad** flags gets set. Therefore, by the fundamental lemma of game playing, we get

$$(1) \le \Pr\left[B^{\mathcal{G}_F(\cdots),\mathcal{V}_F(\cdots),\mathcal{O}_F(\cdot)} \text{ sets } \mathbf{bad}[8-11]\right].$$

Using the adversary B, we shall construct a new adversary C that has access only to the $\mathcal{G}_F$ and $\mathcal{O}_F$ oracles. The adversary C simulates the $\mathcal{V}_F$ oracle in the natural way using its $\mathcal{G}_F$ oracle. We add more flags **bad**$[12-14]$, listed in Table 3, to the description of the $\mathcal{G}_F$ oracle. Clearly we have

$$\Pr\left[B^{\mathcal{G}_F(\cdots),\mathcal{V}_F(\cdots),\mathcal{O}_F(\cdot)} \text{ sets } \mathbf{bad}[8-11]\right] \le \Pr\left[C^{\mathcal{G}_F(\cdots),\mathcal{O}_F(\cdot)} \text{ sets } \mathbf{bad}[8-14]\right],$$

and we see that C plays a random-until-**bad** game. Therefore, we have

$$\Pr\left[C^{\mathcal{G}_F(\cdots),\mathcal{O}_F(\cdot)} \text{ sets } \mathbf{bad}[8-14]\right] \le \frac{\sigma + \cdots + \sigma}{2^n} \le \frac{6\sigma^2}{2^n},$$

rather than $7\sigma^2/2^n$, since one of the σ's gets cancelled out by $(q-1)\sigma = q\sigma - \sigma$ in **bad**$[14]$.

It remains to evaluate the quantity (2). For this, we introduce a random oracle $\mathcal{R}_n$ which returns an n-bit random string upon a query $\left(\overrightarrow{H}, M\right)$. We replace the $\mathcal{G}_F$ oracle with the ideal $\mathcal{R}_n$, as

$$\begin{aligned}\Pr\left[\tilde{B}^{\mathcal{G}_F(\cdots),\mathcal{V}_F(\cdots)} \text{ forges}\right] &= \Pr\left[\tilde{B}^{\mathcal{G}_F(\cdots),\mathcal{V}_F(\cdots)} \text{ forges}\right] - \Pr\left[\tilde{B}^{\mathcal{R}_n(\cdots),\mathcal{V}_F(\cdots)} \text{ forges}\right] \quad (3)\\ &\quad + \Pr\left[\tilde{B}^{\mathcal{R}_n(\cdots),\mathcal{V}_F(\cdots)} \text{ forges}\right]. \quad (4)\end{aligned}$$

By the fundamental lemma of game playing, we see that

$$(3) \le \Pr\left[\tilde{B}^{\mathcal{G}_F(\cdots),\mathcal{V}_F(\cdots)} \text{ sets } \mathbf{bad}[12-14]\right],$$

where the **bad** flags are placed across the two oracles $\mathcal{G}_F$ and $\mathcal{V}_F$. We obtain the following random-until-**bad** games.

$$(3) \leq \Pr\left[\tilde{B}^{\mathcal{G}_F(\cdots),\mathcal{V}_F(\cdots)} \text{ sets } \mathbf{bad}[12-15]\right] \leq \frac{4\sigma^2}{2^n}, \text{ and}$$

$$(4) = \Pr\left[\tilde{B}^{\mathcal{G}_F(\cdots),\mathcal{V}_F(\cdots)} \text{ sets } \mathbf{bad}[15]\right] \leq \frac{\sigma^2}{2^n},$$

which gives us $(6+4+1)\sigma^2/2^n = 11\sigma^2/2^n$ as desired. □

6 Alternative Way of Tag Mixing

Here we mention a variant of BTM. We could use $T \oplus U$ in place of $T \boxplus U$. Then the counter increment is done via the multiplication by $\underline{2}_n$ in the finite field of 2^n elements. This variant has both advantages and disadvantages. After careful consideration, we have decided to choose $T \boxplus U$.

The $T \oplus U$ method does not require the arithmetic addition, which somewhat reduces the size of hardware footprint. Moreover, there exist quite efficient hardware implementations of the multiplication by $\underline{2}_n$. On the other hand, however, the software implementations of the multiplication by $\underline{2}_n$ become a bit costly, depending on the block size n and on the available word size(s) of the platform.

We have observed that the software inefficiency of the $T \oplus U$ method appears to be a little high price to pay for the hardware efficiency. There also exist fairly efficient hardware implementations of the $\boxplus$ operation, and the $\boxplus\underline{1}_n$ increment gains much better software performance on most of the platforms.

7 Improving Security via Tweakable Blockciphers

BTM gives excellent performance but provides security only up to the standard birthday bound. Here we consider the problem of constructing DAE whose security is *beyond* the birthday bound (BBB). The problem was addressed in [7], and BBB constructions are of particular interest if we consider *key wrap* [12], an important application of DAE. With a key-wrap algorithm, one encrypts and authenticates specialized data such as cryptographic keys, where one might desire to ensure the highest security possible. BBB constructions can be used also when one prefers to use a 64-bit blockcipher as the underlying primitive and at the same time ensure security better than the $O(2^{32})$ birthday bound.

The BBB construction described in [7] requires about twelve blockcipher calls to encrypt two blocks of a message, which is hopelessly inefficient and impractical. Here we present a BBB construction that uses a *tweakable* blockcipher [8] as the underlying primitive, instead of using an ordinary blockcipher. Our construction is somewhat more efficient than the one in [7] in a situation where one can start with such a tweakable blockcipher.

Let $\tilde{E}_K : \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$ be a tweakable blockcipher, where $\mathcal{T} = \{0,1\}^n$ is a tweak space. First construct a $2n$-to-$2n$-bit blockcipher E'_{K_1,K_2,K_3} :

$\{0,1\}^{2n} \to \{0,1\}^{2n}$, by using the sENR (simplified Extended Naor-Reingold) construction [11], where K_1 and K_2 are the keys for the underlying tweakable blockcipher, and $K_3 \in \{0,1\}^n$. Recall that one call to E' requires one multiplication over $GF(2^n)$ and two calls to $\tilde{E}$. Then construct the BTM mode having a block size of $2n$ bits, using E' as its underlying blockcipher. This construction ensures security beyond the $O(2^{n/2})$ bound. The construction is not too inefficient; to encrypt two blocks of a message, it requires about one multiplication over $GF(2^{2n})$, one multiplication over $GF(2^n)$, and two calls to $\tilde{E}$.

Unfortunately, the construction still has the following problems.

1. The key length is more than n bits; the key space of E' is rather large.
2. The tag size is $2n$ bits instead of n bits; the ciphertext is somewhat long.

It remains open to provide a BBB construction (based on a tweakable blockcipher) which resolves the two problems. Also, our unorthodox method involves using a tweakable blockcipher as the underlying primitive, which itself must have BBB security. This implies that the standard construction of a tweakable blockcipher in [8] is not suitable for our purpose. Although the constructions in [5,4,11] stand as potential candidates for our $\tilde{E}$, there is no known construction that completely fulfills our requirements. The basic problem of designing an efficient tweakable blockcipher with BBB security, possibly from scratch, still remains to be solved.

Acknowledgments

The authors would like to express their thanks to the anonymous reviewers of SAC 2009 for their helpful comments.

References

1. Bellare, M., Kilian, J., Rogaway, P.: The security of the cipher block chaining message authentication code. J. Comput. Syst. Sci. 61(3), 362–399 (2000)
2. Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 531–545. Springer, Heidelberg (2000)
3. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 409–426. Springer, Heidelberg (2006)
4. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The Skein hash function family. Submission to NIST (2008), `http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/submissions_rnd1.html`
5. Goldenberg, D., Hohenberger, S., Liskov, M., Schwartz, E.C., Seyalioglu, H.: On tweaking Luby-Rackoff blockciphers. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 342–356. Springer, Heidelberg (2007)
6. Iwata, T., Kurosawa, K.: OMAC: One-key CBC MAC. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 129–153. Springer, Heidelberg (2003)

7. Iwata, T., Yasuda, K.: HBS: A single-key mode of operation for deterministic authenticated encryption. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 394–415. Springer, Heidelberg (2009)
8. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 31–46. Springer, Heidelberg (2002)
9. Luby, M., Rackoff, C.: How to construct pseudorandom permutations from pseudorandom functions. SIAM J. Comput. 17(2), 373–386 (1988)
10. McGrew, D.A., Viega, J.: The security and performance of the Galois/counter mode (GCM) of operation. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 343–355. Springer, Heidelberg (2004)
11. Minematsu, K.: Beyond-birthday-bound security based on tweakable block cipher. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 308–326. Springer, Heidelberg (2009)
12. NIST: AES key wrap specification (2001)
13. NIST: Recommendation for block cipher modes of operation: The CMAC mode for authentication (2005)
14. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: A block-cipher mode of operation for efficient authenticated encryption. In: ACM CCS, pp. 196–205. ACM Press, New York (2001)
15. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 373–390. Springer, Heidelberg (2006)
16. Whiting, D., Housley, R., Ferguson, N.: Counter with CBC-MAC (CCM). Submission to NIST (2002), `http://csrc.nist.gov/groups/ST/toolkit/BCM/index.html`

A Proof of Lemma 1

We have

$$\begin{aligned}
&\Pr\bigl[G(L,U)=\underline{0}_n \bigm| L,U \xleftarrow{\$} \{0,1\}^n\bigr]\\
=&\sum_{U_0\in\{0,1\}^n} \Pr\bigl[U=U_0 \wedge G(L,U)=\underline{0}_n \bigm| L,U \xleftarrow{\$} \{0,1\}^n\bigr]\\
=&\sum_{U_0\in\{0,1\}^n} \Pr\bigl[U=U_0 \bigm| U \xleftarrow{\$} \{0,1\}^n\bigr]\cdot\Pr\bigl[G(L,U_0)=\underline{0}_n \bigm| L \xleftarrow{\$} \{0,1\}^n\bigr].
\end{aligned}$$

Now put $x := \deg_L G$ and let $Z \subset \{0,1\}^n$ be the set of U_0 which makes the coefficient of L^x zero. Note that we have $|Z| \le \deg_L^U G$. We get

$$\begin{aligned}
&\sum_{U_0\in Z} \Pr\bigl[U=U_0 \bigm| U \xleftarrow{\$} \{0,1\}^n\bigr]\cdot\Pr\bigl[G(L,U_0)=\underline{0}_n \bigm| L \xleftarrow{\$} \{0,1\}^n\bigr]\\
&\qquad+\sum_{U_0\notin Z} \Pr\bigl[U=U_0 \bigm| U \xleftarrow{\$} \{0,1\}^n\bigr]\cdot\Pr\bigl[G(L,U_0)=\underline{0}_n \bigm| L \xleftarrow{\$} \{0,1\}^n\bigr]\\
\le&|Z|\cdot\frac{1}{2^n}\cdot 1+\sum_{U_0\notin Z}\frac{1}{2^n}\cdot\frac{\deg_L G}{2^n}\\
\le&\frac{\deg_L^U G+\deg_L G}{2^n},
\end{aligned}$$

as desired.

B Computing the Probabilities in Table 2

In the following computations we introduce a variable

$$\mu := \max\{h_0, \ldots, h_{d-2}, m\}.$$

In other words, μ denotes the maximum length of a component in the vector $\big(\overrightarrow{H}, M\big)$. We also introduce a new variable

$$\lambda := h_0 + \cdots h_{d-2} + m.$$

The superscript (i) denotes the fact that the variable comes from the i-th query.

We compute the probabilities one by one. We start with $\mathbf{bad}[0]$. This event corresponds to outputting the hash key L. From Lemma 1 we have

$$\begin{aligned}\Pr\big[\mathbf{bad}[0]\big] &\le \sum_{i=1}^{q} \Pr\Big[F_{L,U}\big(H_0^{(i)}, \ldots, H_{d^{(i)}-2}^{(i)}, M^{(i)}\big) = \underline{0}_n\Big] \\ &\le \sum_{i=1}^{q} \frac{d^{(i)} - 1 + \mu^{(i)}}{2^n} \le \sum_{i=1}^{q} \frac{\lambda^{(i)}}{2^n},\end{aligned}$$

which is then bounded by $\sigma/2^n$.

The probability of $\mathbf{bad}[1]$ can be done in the exactly same manner, as this corresponds to outputting the other hash key U. We have $\Pr\big[\mathbf{bad}[1]\big] = \Pr\big[\mathbf{bad}[0]\big] \le \sigma/2^n$.

The event $\mathbf{bad}[2]$ means a collision among the hash values. Using Lemma 1, we compute

$$\begin{aligned}\Pr\big[\mathbf{bad}[2]\big] &\le \sum_{i=2}^{q}\sum_{j=1}^{i-1} \Pr\Big[F_{L,U}\big(\overrightarrow{H}^{(i)}, M^{(i)}\big) = F_{L,U}\big(\overrightarrow{H}^{(j)}, M^{(j)}\big)\Big] \\ &\le \sum_{i=2}^{q}\sum_{j=1}^{i-1} \frac{\max\big\{d^{(i)} - 1, d^{(j)} - 1\big\} + \max\big\{\mu^{(i)}, \mu^{(j)}\big\}}{2^n} \\ &\le (q-1)\sum_{i=1}^{q} \frac{d^{(i)} - 1 + \mu^{(i)}}{2^n} \le (q-1)\sum_{i=1}^{q} \frac{\lambda^{(i)}}{2^n},\end{aligned}$$

which is then bounded by $(q-1)\sigma/2^n$.

We proceed to $\mathbf{bad}[3]$. We use Lemma 1 to get

$$\begin{aligned}\Pr\big[\mathbf{bad}[3]\big] &\le \sum_{i=2}^{q}\sum_{j=1}^{i-1}\sum_{\beta=0}^{m^{(j)}-1} \Pr\Big[F_{L,U}\big(H_0^{(i)}, \ldots, H_{d^{(i)}-2}^{(i)}, M^{(i)}\big) = T^{(j)} \boxplus U \boxplus \underline{\beta}_n\Big] \\ &\le \sum_{i=2}^{q}\sum_{j=1}^{i-1}\sum_{\beta=0}^{m^{(j)}-1} \frac{d^{(i)} - 1 + \mu^{(i)}}{2^n} \\ &\le \sum_{i=2}^{q}\sum_{j=1}^{i-1}\sum_{\beta=0}^{m^{(j)}-1} \frac{\lambda^{(i)}}{2^n} = \sum_{i=2}^{q} \frac{\lambda^{(i)}}{2^n} \sum_{j=1}^{i-1}\sum_{\beta=0}^{m^{(j)}-1} 1 = \sum_{i=2}^{q} \frac{\lambda^{(i)}}{2^n} \sum_{j=1}^{i-1} m^{(j)},\end{aligned}$$

which we can bound as $\sigma/2^n \cdot \sigma = \sigma^2/2^n$.

The quantity $\Pr\big[\mathbf{bad}[4]\big]$ can be evaluated relatively easily. We get

$$\Pr\big[\mathbf{bad}[4]\big] \le \sum_{i=2}^{q} \sum_{\alpha=0}^{m^{(i)}-1} \Pr\Big[T^{(i)} \boxplus U \boxplus \underline{\alpha}_n = \underline{0}_n\Big] \le \sum_{i=2}^{q} \frac{m^{(i)}}{2^n},$$

which must be less than $\sigma/2^n$.

The event $\Pr\big[\mathbf{bad}[5]\big]$ can be treated in a way similar to $\Pr\big[\mathbf{bad}[4]\big]$. We have $\Pr\big[\mathbf{bad}[5]\big] = \Pr\big[\mathbf{bad}[4]\big] \le \sigma/2^n$.

Also, the quantity $\Pr\big[\mathbf{bad}[6]\big]$ is exactly the same as $\Pr\big[\mathbf{bad}[3]\big]$. We have $\Pr\big[\mathbf{bad}[6]\big] = \Pr\big[\mathbf{bad}[3]\big] \le \sigma^2/2^n$.

Lastly, we go on to treat the event $\mathbf{bad}[7]$. We have

$$\begin{aligned}
\Pr\big[\mathbf{bad}[7]\big] &\le \sum_{i=2}^{q} \sum_{j=1}^{i-1} \Pr\Big[\bigvee_{\alpha=0}^{m^{(i)}-1} \bigvee_{\beta=0}^{m^{(j)}-1} \big(T^{(i)} \boxplus U \boxplus \underline{\alpha}_n = T^{(j)} \boxplus U \boxplus \underline{\beta}_n\big)\Big] \\
&\le \sum_{i=2}^{q} \sum_{j=1}^{i-1} \Pr\Big[\bigvee_{\gamma=-m^{(i)}+1}^{m^{(j)}-1} \big(T^{(i)} = T^{(j)} \boxplus \underline{\gamma}_n\big)\Big] \\
&\le \sum_{i=2}^{q} \sum_{j=1}^{i-1} \frac{m^{(i)} + m^{(j)} - 2}{2^n} \\
&\le (q-1) \sum_{i=1}^{q} \frac{2m^{(i)} - 2}{2^n},
\end{aligned}$$

which is less than $2(q-1)(\sigma-q)/2^n$. This concludes the computation of the probabilities in Table 2.

C Computing the Probabilities in Table 3

Again we use the variable

$$\mu := \max\{h_0, \ldots, h_{d-2}, m\}.$$

As usual, the superscript (i) denotes the fact that the variable comes from the i-th query that the adversary makes.

We begin with $\Pr\big[\mathbf{bad}[8]\big]$. We get

$$\Pr\big[\mathbf{bad}[8]\big] \le \sum_{r=1}^{\sigma-q} \Pr\big[W^{(r)} \boxplus U = \underline{0}_n\big] \le \sum_{r=1}^{\sigma-q} \frac{1}{2^n},$$

which must be bounded by $(\sigma-q)/2^n$.

The event $\mathbf{bad}[9]$ can be treated in a similar way. We obtain $\Pr\big[\mathbf{bad}[9]\big] = \Pr\big[\mathbf{bad}[8]\big] \le (\sigma-q)/2^n$.

Next we evaluate the probability $\Pr\big[\mathbf{bad}[10]\big]$. Using Lemma 1, we compute as

$$\begin{aligned}\Pr\big[\mathbf{bad}[10]\big] &\le \sum_{r=1}^{\sigma-q}\sum_{j=1}^{q}\Pr\Big[W^{(r)} \boxplus U = F_{L,U}\big(H_0^{(j)},\ldots,H_{d^{(j)}-2}^{(j)},M^{(j)}\big)\Big] \\ &\le \sum_{r=1}^{\sigma-q}\sum_{j=1}^{q}\frac{\max\{d^{(j)}-1,1\}+\mu^{(j)}}{2^n},\end{aligned}$$

which can be bounded by $(\sigma-q)(\sigma+q)/2^n$.

The probability $\Pr\big[\mathbf{bad}[11]\big]$ is exactly the same as $\Pr\big[\mathbf{bad}[10]\big]$. We have $\Pr\big[\mathbf{bad}[11]\big] = \Pr\big[\mathbf{bad}[10]\big] \le (\sigma-q)(\sigma+q)/2^n$.

The events $\mathbf{bad}[12]$, $\mathbf{bad}[13]$ and $\mathbf{bad}[14]$ can be treated in ways similar to $\mathbf{bad}[0]$, $\mathbf{bad}[1]$ and $\mathbf{bad}[2]$, respectively, whose probabilities have been already computed in Appendix B. We get

$$\begin{aligned}\Pr\big[\mathbf{bad}[12]\big] &= \Pr\big[\mathbf{bad}[0]\big] \le \frac{\sigma}{2^n}, \\ \Pr\big[\mathbf{bad}[13]\big] &= \Pr\big[\mathbf{bad}[1]\big] \le \frac{\sigma}{2^n}, \\ \Pr\big[\mathbf{bad}[14]\big] &= \Pr\big[\mathbf{bad}[2]\big] \le \frac{(q-1)\sigma}{2^n},\end{aligned}$$

as expected.

Lastly, we handle $\mathbf{bad}[15]$. Observe that this event is nothing but a forgery by making random guesses. So we obtain

$$\Pr\big[\mathbf{bad}[15]\big] \le \sum_{i=1}^{q}\Pr\big[\mathcal{V}_F(\cdots) \ne \perp\big] \le \frac{q}{2^n}.$$

Thus we have completed computing the probabilities in Table 3.