Generating short-output digest functions

L.H. Nguyen and A.W. Roscoe
Oxford University Computing Laboratory
E-mail addresses: {Long.Nguyen, Bill.Roscoe@comlab.ox.ac.uk}
This paper forms the sixth chapter of my thesis, and reports my recent research result towards the first goal of my proposal: "Efficient and secure digest functions".

This paper has also been submitted to AFRICACRYPT 2010 in January 2010.

Abstract

This paper introduces two related methods of generating a new cryptographic primitive termed *digest* which has similarities to ϵ-balanced and almost universal hash functions. Digest functions, however, typically have a very short output, e.g. 16-64 bits, and hence they are not required to resist collision and inversion attacks. They also have the potential to be very fast to compute relative to long-output hash functions. The first construction uses Toeplitz matrix multiplication, which is similar to a Toeplitz based universal hashing algorithm of Krawczyk, whose security requirements can be reduced to the underlying ϵ-biased sequences of random variables. The second is based on integer multiplications which have, perhaps surprisingly, a similar structure to Toeplitz matrix multiplication. However, due to the complication of carry bits, a rigorous mathematical proof of the second construction cannot be provided. We instead exploit the short output of digest functions to carry out statistical analysis, including chi-square tests, quantile-quantile plots and maximum median calculation, of digest collision and distribution test results to argue for the security of the second construction.

1 Introduction

In this paper, we investigate the design, constructions and security of a new cryptographic primitive termed (ϵ-balanced) *digest function*, which was recently introduced in [**?**]. Digest functions have similarities to cryptographic hash and universal hash functions, the second of which was extensively studied in the early nineties by many researchers [**?**, **?**, **?**, **?**, **?**]. The main difference between digest functions and nearly all other families of hash functions is that digest functions typically have a very short output in the range between 16 and 64 bits. For this reason, it is not possible to generate digest functions which can resist collision and inversion attacks in the same way as cryptographic hash functions. This feature however opens the way for efficiently computed digest functions arising from both their short output and the potential of parallel computation. We observe that parallelism is unfortunately not available with many current cryptographic hash functions, such as SHA-I and MD5, whose design strictly follows the *sequential* Merkle-Damgård construction [**?**].[1] The main goal of this

[1]There have been a number of newly proposed cryptographic hash functions, such as MD6 [**?**] and SANDstorm [**?**] (both of which were entered into the hash competition [**?**] organised by NIST recently), which make use of the hash-tree structure introduced by Merkle [**?**] to benefit from parallel computation. We note that there are two weaknesses of a hash-tree construction: (1) the speed-up factor of parallel computation is

paper is therefore to study efficient digest constructions which are provably secure regarding their security properties, including distribution and some other imposed restrictions in a digest function, e.g. the likelihood of digest collision with respect to distinct keys are taken into consideration.

The specification of a short-output digest function arose from a new (and non-standard) family of authentication protocols [**?**, **?**, **?**, **?**] which can create secure communication from human trust and human interactions to overcome the needs for passwords as in Bluetooth and security certificates as in PKI. In this new authentication technology, to complete a protocol run, devices' human owners have to manually compare a short digest of the information they want to authenticate. It is therefore desirable to design efficient and secure short-output digest functions which can be implemented in small devices and lightweight cryptographic applications. Another application of digest functions is a new family of digital signature schemes where the use of a cryptographic hash function is replaced by digest to potentially increase efficiency. A typical example of this new type of digital signature is Flexi-MAC [**?**] whose single certificate can be checked to various degrees, depending on the perceived security threat, the time and computing power available.

Although short-output digest functions can be computed by just truncating the first small number of bits of a cryptographic or universal hash function, as discussed in Section **??**, we will point out that this strategy does not exploit the short output of digest and parallel computation to increase computational efficiency. We subsequently prove that a well-studied universal hashing algorithm based on Toeplitz matrix multiplication and ϵ-biased sequences of random variables can generate digests with the properties we require in Definition **??**.

The second main contribution of this paper is the discovery of a structural similarity between Toeplitz matrix multiplication and the "school book" algorithm for integer long multiplication. Even though these are not exactly the same due to the effect of carry bits in integer multiplications, this work suggests another method using word multiplications which can be computed efficiently on any processor. We note that carry bits in integer multiplication make it not feasible to give a mathematical proof of digest properties, we will instead exploit the short output of digest functions to facilitate our digest collision and distribution tests with statistical analysis. This approach includes computing the p-values in chi-square tests, quantile-quantile plots and the maximum median calculation of collision and distribution test results on word multiplication based digest algorithm. This will suggests carry bits in integer long multiplication do not introduce much bias into the digest computation regime relative to the Toeplitz based method. To the best of our knowledge, this is the first time when statistical approach has been used to analyse the security properties of hash, universal hash and digest functions. Without carry bits our word multiplication based digest construction is identical to the Toeplitz one, and indeed such a carry-less instruction for word multiplication has been introduced into the next generation of Intel processor [**?**].

Both our digest constructions in this paper can benefit from parallel computation should they be computed in multi-core microprocessors, such as a 48-core CPU recently announced by Intel [**?**].

---

bounded above by the depth of the hash tree, i.e. $\log n$ where n is the length of input messages, relative to the sequential Merkle-Damgård construction; and (2) to the best of our knowledge, there might be security weaknesses due to the variable depth of the tree arising from variable length input message, and thus this structure has not been widely adopted to construct efficient hash functions for commercial purposes.

2 Notation

Every operator used in this paper is bitwise, which is equivalent to working over the finite field $\mathbb{F}_2$. Therefore, we will replace bitwise exclusive-OR and bitwise AND with addition (or summation) and multiplication in $\mathbb{F}_2$, respectively. We note that addition is equivalent to subtraction in $\mathbb{F}_2$, and so we will stick to the use of addition to simplify the notation. Also, to avoid confusion, '$\times$' and '*' are used to indicate finite field multiplication and respectively integer multiplication.

In this paper, we only consider non-zero messages representable by M bits, and hence the set of all messages is denoted $X = \{1...(2^M - 1)\}$. K, M and b are the bitlengths of digest or hash key, input message, and digest or hash output.

Definition 1 [**?**] An ϵ-balanced universal hash function is a set of 2^K hash functions $h_k()$ where $k \in \{0...(2^K-1)\}$ such that $h_k : X \to Y$; here $X = \{1...(2^M-1)\}$ and $Y = \{0...(2^b-1)\}$. Moreover, for every non-zero message m and every hash output $y \in Y$, we have:

$$\Pr_{\{k \in R\}}[digest(k, m) = y] \leq \epsilon$$

Extending the definition of an ϵ-balanced universal hash function, we define an ϵ-balanced digest function as follows.[2]

Definition 2 [**?**] A ϵ-balanced *digest* function: $digest : R \times X \to Y$ where $R = \{0...(2^K - 1)\}$, $X = \{1...(2^M - 1)\}$ and $Y = \{0...(2^b - 1)\}$ are the set of all keys, input messages and digest outputs, and moreover:

- for every $m \in X$ and $y \in Y$, $\Pr_{\{k \in R\}}[digest(k, m) = y] \leq \epsilon$
- for every $m, m' \in X$ $(m \neq m')$ and any $\theta \in R$: $\Pr_{\{k \in R\}}[digest(k, m) = digest(k + \theta, m')] \leq \epsilon$

Note that any function satisfying Definition **??** also satisfies both Definition **??** and the requirements of an ϵ-almost universal hash function defined in [**?**, **?**].

Definition 3 [**?**, **?**] Let S be a distribution of sequences of length l. Let (α, s) denote the scalar (or inner) product modulo 2 of $\alpha \in \{0,1\}^l$ and $s \in \{0,1\}^l$.

- S is said to pass the linear test $\alpha \in \{0,1\}^l$ with bias ϵ if $|\Pr_{\{s \in S\}}[(\alpha, s) = 1] - 1/2| \leq \epsilon$.
- S is said to be an ϵ-biased distribution if it passes all linear tests $\alpha \neq 0$ with bias ϵ.

Definition 4 A Toeplitz or diagonal-constant matrix A is a (not necessary square) matrix where each descending diagonal from left to right is constant, i.e. for all pairs of indexes (i, j): $A_{i,j} = A_{i+1,j+1}$.

[2] In the second requirement of a digest function, we can replace $+\theta$ with $\oplus\theta$ since the bitwise exclusive-or operator is equivalent to addition over the finite field $\mathbb{F}_2$.

3 Background information

It would be a great mistake to compute $digest(k, m)$ as some function $f()$ of k and some similar length $digest'(m)$, i.e. $digest'()$ is not collision-resistant or the lengths of $digest'$ and $digest$ are similar, which it might be tempting to do.

$$digest(k, m) = f(k, digest'(m))$$

This is because an intruder could search for a set of different input messages all digesting to the same value $digest'(m)$ irrespective of the value of k, i.e. any pair m and m' of these would violate the second requirement of a digest function. We therefore need to embed the key k fundamentally into the calculation of $digest(k, m)$: it cannot be an afterthought.

Although digest output is short, it is challenging to construct digest algorithms that have both efficiency and provable security relative to the digest's requirements. Thus the majority of short-output digest constructions invented to date [**?**, **?**, **?**] make use of cryptographic hash functions, as seen in the following examples.

Pasini and Vaudenay [**?**], Gehrmann et al.[**?**], and the Bluetooth white-paper [**?**] suggested the following scheme:

$$digest(k, m) = trunc_b\left(hash(k \parallel m)\right)$$

where $trunc_b()$ function truncates to the leading b bits. We make two observations about this.

1. The definition of a cryptographic hash function does not normally specify the distribution of individual groups of bits, and so whether or not this is a good idea will very much depend on the properties we require of the hash function and the properties we want from the digest function. It follows that a specific analysis would be required for any particular standard cryptographic hash function proposed.

2. Computing a longhash operating on long words as in the sequential Merkle-Damgård construction can be potentially expensive, i.e. neither does the Merkle-Damgård structure permit parallelism nor exploit the short output of a digest function to increase efficiency.

Taking a different approach, Gehrmann and Nyberg [**?**] proposed using error-correcting codes, such as the extended Reed-Solomon codes, to construct the digest function. This has the advantage of having a coherent mathematical structure. On the other hand, to have unconditional security, the algorithm limits the bitlength of the input message to some fixed number, such as 128 or 256, which is the length of a dataword input by the algorithm. To digest any significant amount of data, the algorithm first compresses the input message into that number of bits by using a cryptographic hash which can be inefficient as discussed above. This feature also makes the scheme be potentially vulnerable to attacks should the intruder find (off line) a collision on the cryptographic hash, i.e. this is equivalent to finding a pair of distinct messages which violate the second requirement of a digest function. The reason for this weakness is because the input message is not entirely linked to the key in digest computation, as discussed at the beginning of this section.

4 Toeplitz matrix product construction of a digest function

We now describe a universal hash function construction using Toeplitz matrixes whose binary elements are drawn from ϵ-biased sequences of (pseudo)random bits. This construction was

proved by Krawczyk in [**?**] to satisfy the requirements of an ϵ-balanced universal hash function. We will then show the *same* algorithm can be used to generate a digest function with the properties we require.

Suppose we want to compute a b-bit universal hash of a (non-zero) M-bit message m, then the key k must be drawn from an ϵ-biased distribution R of length $K = M + b - 1$. Using the key k, we can generate a Toeplitz matrix $A(k)$ of M rows and b columns. Using matrix product, we define:

$$h_k(m) \quad = \quad m \times A(k) \tag{1}$$

The symbol '$\times$' represents the product of the vector m and the matrix $A(k)$ in $\mathbb{F}_2$.

Theorem 1 [**?**] $h_k(m)$ defined in Equation (**??**) above satisfies the definition of an ϵ-balanced universal hash function.

Readers who are interested in the proof of this theorem can find it in the paper of Krawczyk [**?**] (Theorem 9, Lemma 10 and Theorem 5). The proof makes use of a known relation between Discrete Fourier Transform of matrix multiplication and ϵ-biased distributions.

If we replace a universal hash function $h_k(m)$ with a digest function $digest_T(k, m)$, where index T indicates a Toeplitz based digest construction, we have:

$$digest_T(k, m) \quad = \quad m \times A(k) \tag{2}$$

Theorem 2 $digest_T(k, m)$ defined in Equation (**??**) above satisfies the definition of an ϵ-balanced digest function.

Proof The first requirement of an ϵ-balanced digest function is satisfied thanks to Theorem **??**.

An observation on the digest construction using Toeplitz matrix multiplication is that this construction is linear in terms of both input message m and key k. The latter is true because for any k and θ of length $K = M + b - 1$ bits, we always have $A(k+\theta) = A(k) + A(\theta)$ and hence $digest_T(k + \theta, m) = digest_T(k, m) + digest_T(\theta, m)$.

For any pair of distinct messages (m, m') and any θ of length K bit, a digest collision $digest_T(k, m) = digest_T(k + \theta, m')$ is equivalent to:

$$\begin{aligned}
digest_T(k, m) &= digest_T(k + \theta, m') \\
m \times A(k) &= m' \times A(k + \theta) \\
m \times A(k) &= m' \times A(k) + m' \times A(\theta) \\
(m - m') \times A(k) &= m' \times A(\theta) \\
digest_T(k, m - m') &= m' \times A(\theta)
\end{aligned}$$

Since we have fixed m, m' and θ, let us denote $m'' = m - m'$ and $\alpha = m' \times A(\theta)$. And the second requirement of a digest function is satisfied because:

$$\Pr_{\{k \in R\}}[digest_T(k, m) = digest_T(k + \theta, m')] = \Pr_{\{k \in R\}}[digest_T(k, m'') = \alpha] \leq \epsilon$$

The inequality in the above equation is true thanks to the first requirement of a digest function (or Theorem **??**). ∎

In the above Toeplitz based digest computation, the key k is assumed to have a bitlength of $K = M + b - 1$ which can be long if the input message is large. In practice, however, a key is normally of the size of a typical cryptographic hash function, say $r = 160$ or 256 bits, and hence we need to derive the required number K of bits pseudorandomly from the initial r bits.

One possible way is to use a Linear Feedback Shift Register (LFSR) as suggested by Alon et al. [**?**] (Proposition 1 of Section 3). In this LFSR construction, the first $r/2$ bits of key k will be used to select the linear structure of the underlying $r/2$-bit LFSR used, i.e. this is equivalent to selecting an irreducible polynomial of degree $r/2$. This LFSR is then seeded by the other $r/2$ bits of key k. This construction has been shown by Alon et al. [**?**] to produce an ϵ-biased distribution on length $K = M + b - 1$ with $\epsilon = \frac{K}{2^{r/2}}$, where each sequence is generated out of r initial bits.

There are a number of other well-studied constructions of ϵ-biased sequences due to Alon et al. [**?**], including Legendre symbol construction and scalar product construction, both of which can be found in their paper.

4.1 Efficiency of Toeplitz matrix based digest functions

Given an ϵ-biased sequence of length $K = M + b - 1$ bits,[3] it is straightforward to design a customised hardware implementation of $digest_T(k, m)$, which can also benefit from parallelism.

We first look at the complexity of a sequential implementation of this algorithm. The cost of multiplying a M-bit vector by a $M \times b$ matrix is $2Mb$ bit operations, which are either AND or XOR in this case. On a w-bit microprocessor, we can expect this to take at least $2Mb/w$ bitwise word operations. This figure clearly shows that the cost of computing digest function is at least proportional to its output length. To simplify the calculation, we assume that w equals b, and so this is equivalent to $2M$ word operations. Since the latency of most logical instructions (AND, OR, NOT, and XOR) of modern microprocessors is 1 clock cycle, executing $2M$ bitwise word operations takes $2M$ arithmetic clock cycles.

As regards parallel computation, we observe that a matrix multiplication permits parallelism in two distinct ways. First, each digest output bit can be computed on its own, and hence the complexity can be improved by a factor of b. Secondly, a $M \times b$ matrix multiplication can be split into $t = M/w$ parallel $w \times b$ matrices multiplications with their resulting b-bit vectors being XORed to produce the digest value. The second method is thus $t = M/w$ times more efficient than the sequential computation.

5 Comparing Toeplitz matrix and integer long multiplication

In this section, we will show a structural similarity between the "school book" algorithm for integer long multiplication and Toeplitz matrix multiplication. This similarity, in the next section, will lead to a more efficient algorithm for digest function.

Suppose as in the Toeplitz-based method of Section **??**, key k is drawn from an ϵ-biased distribution R of length $K = M + b - 1$. Consider a "school book" algorithm for integer long multiplication of two numbers k and m represented in binary, i.e. $k = (k_1 \dots k_K)$ and

[3] The K-bit ϵ-biased sequence can be derived (off-line) pseudorandomly from a shorter key and stored in both devices for subsequent digest computation.

$m = (m_1 \dots m_M)$. The following is, however, only a part of the table that is used to carry out the integer long multiplication.

	k_1	k_2	$\cdots$	k_{b-1}	k_b	$\cdots$	k_K
					m_1	$\cdots$	m_M
	$m_M \times k_1$	$m_M \times k_2$	$\cdots$	$m_M \times k_{b-1}$	$m_M \times k_b$	$\cdots$	$\cdots$
$\cdots$	$m_{M-1} \times k_2$	$m_{M-1} \times k_3$	$\cdots$	$m_{M-1} \times k_b$	$m_{M-1} \times k_{b+1}$	$\cdots$	
$\cdots$	$m_{M-2} \times k_3$	$m_{M-2} \times k_4$	$\cdots$	$m_{M-2} \times k_{b+1}$	$m_{M-2} \times k_{b+2}$	$\cdots$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
$\cdots$	$m_1 \times k_M$	$m_1 \times k_{M+1}$	$\cdots$	$m_1 \times k_{K-1}$	$m_1 \times k_K$		
$\cdots$	d_1	d_2	$\cdots$	d_{b-1}	d_b	$\cdots$	$\cdots$

We are only interested in the overlap of the expanded multiplication, where each bit of the resulting product $(d_1 \dots d_b)$ is influenced *directly* by every bit of the message m. In other words, each carry bit c_i which is instrumental in the computation of d_i is considered to have an *indirect* impact.

We note that the overlapping part of the expanded multiplication can be interpreted as a matrix of b columns and M rows. Moreover, the bits k_is in each row of this matrix are shifted by 1 position to the right hand side as we move from one to the upper row. This means that if we reverse the order of these rows in the expanded multiplication, we will have the property of a Toeplitz matrix. As a result, the b-bit digest value is equivalent to:

$$digest_M(k, m) \quad = \quad m \times A(k) + \ (c_1 \dots c_b) \tag{3}$$

Note that the index "M" in $digest_M(k, m)$ indicates an integer long multiplication based digest construction. Here $A(k)$ is a Toeplitz matrix of b columns and M rows, which is constructed by the same $K = M + b - 1$ bits of (or derived from) key k, and $(c_1 \dots c_b)$ is a bit vector whose elements are carry bits which are instrumental in the computation of the corresponding output or digest bits.

The above description demonstrates a structural similarity between integer long multiplication and Toeplitz matrix multiplication, however they are not equivalent due to the impact of carry bits which are accumulated from d_b all the way to d_1 in an integer long multiplication. Nevertheless, it opens the way for devising new digest constructions using word multiplications which can be computed fast on any microprocessor. This will be addressed in the next section.

6 Word multiplication based digest functions

Although the Toeplitz matrix based digest function can be computed efficiently on customised hardware, many applications of digest functions will need to carry out this operation in software on standard and sometimes basic microprocessors. In this section, we will show that another digest function $digest_{WM}(k, m)$ with strong structural similarities to both $digest_T(k, m)$ and $digest_M(k, m)$ can be calculated using standard integer multiplication for half or whole word blocks that are implemented efficiently in just about all microprocessors. This method is inspired by the relation between the "school book" algorithm for integer long multiplication and Toeplitz matrix multiplication, as demonstrated in Section **??**.

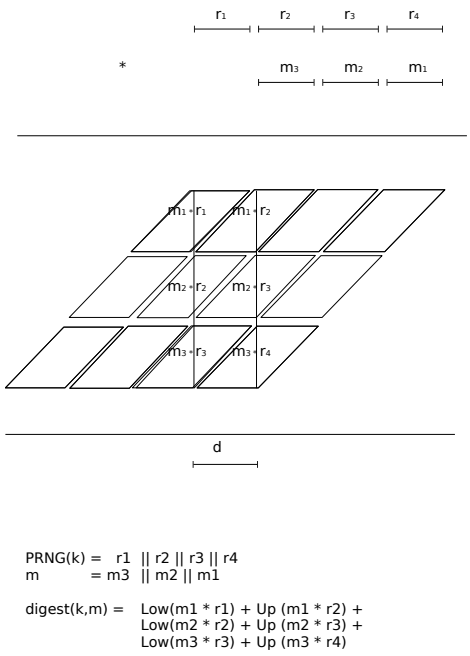


Figure 1: Word multiplication model $digest_{WM}(k, m)$. Each parallelogram equals the expansion of a word multiplication between a b-bit key block and a b-bit message block.

Let us divide message m into b-bit blocks $[m_1, \ldots, m_{t=\frac{M}{b}}]$. We generate $(t+1)$ pseudorandom b-bit blocks r_i derived from key k.[4] In the following equation, the index "WM" of $digest_{WM}(k, m)$ indicates a b-bit word multiplication based digest function.

$$digest_{WM}(k, m) \quad = \quad \sum_{i=1}^{t} [\mathrm{Low}(m_i * r_i) + \mathrm{Up}(m_i * r_{i+1})] \tag{4}$$

Here, * refers to a word multiplication of two b-bit blocks which produces a $2b$-bit output. The output of each word multiplication can be partitioned into the lower and upper (b-bit) halves by applying Low() and Up() functions respectively. Both '+' and $\sum$ used in Equation (**??**) above are additions in $\mathbb{F}_2$.

To see why $digest_{WM}(k, m)$ resembles $digest_M(k, m)$, we give a simple example when the bitlength of the input message is $M = tb = 3b$ in Figure **??**.

As was pointed out in Section **??**, there may be some asymmetry due to carry bits in integer or word multiplications. However, the effect of carry bits in this construction is reduced significantly relative to an integer long multiplication of $digest_M(k, m)$ as described in Section **??**, where carry bits are accumulated from the least to the most significant bits of the product. In contrast, by partitioning an integer long multiplication into multiple (b-bit) word multiplications, carry bits are only accumulated locally, i.e. up to $2b - 1$ bits to the left hand side of each position as opposed to $M(M + b - 1)$ bits in an integer long multiplication of Section **??**.

[4]Any pseudorandom number generator (PRNG) which produces ϵ-biased sequences can be used here. A possible candidate is a LFSR whose linear structure and initial seed are derived from key k as described in Section **??**.

The above construction produces a b-bit digest, in practice, we might want to construct longer digests whose output is a multiple of b, i.e. xb-bit digest function. Using the same idea, it is not hard to generalise the above algorithm to multiple-word digest construction as follows.

We still divide m into b-bit blocks $[m_1, \ldots, m_{t=\frac{M}{b}}]$. However, we will need to generate $t + x$ pseudorandom b-bit blocks r_i derived from key k to compute a xb-bit digest. For all $j \in \{1...x\}$, we then define:

$$d_j = \sum_{i=1}^{t} [\text{Low}(m_i * r_{i+j-1}) + \text{Up}(m_i * r_{i+j})]$$

And

$$digest_{WM}(k, m) = (d_1 \cdots \ d_x)$$

Note that the negative and biased impact of carry bits in integer multiplications can be eliminated entirely if we make use of the carry-less multiplication instruction, termed PCLMULQDQ in the next generation of Intel processors [**?**], that computes the carry-less multiplication of two 64-bit operands without the generation or propagation of carry bits.[5] This will make $digest_{WM}(k, m)$ identical to $digest_T(k, m)$ as defined in Equation (**??**) of Section **??**.

6.1 Efficiency of word multiplication based digest functions

Given an ϵ-biased sequence of length $K = M + b - 1$, the computation of a $(b = 64)$-bit $digest_{WM}(k, m)$ as defined in Equation (**??**) consists of $2t = 2M/b$ word multiplications and $2t = 2M/b$ bitwise XOR operations.[6] In the worst case, each word multiplication takes $w = b$ clock cycles to complete, but it is always significantly faster in many modern microprocessors which have RISC pipeline instructions. For example, AMD (K8 and K10) and Intel CPUs [**?**] can dispatch a $64 \times 64 = 128$-bit MUL once every 2-4 cycles with a latency of 4-7 cycles. To simplify the calculation, we only consider non-pipeline execution, and thus the number of *arithmetic* clock cycles of computing a (b=64)-bit $digest_{WM}(k, m)$ in an AMD CPU is $\frac{2M}{64}4 + \frac{2M}{64} = 5M/32$ compared to $2M$ arithmetic clock cycles of computing $digest_T(k, m)$ of Section **??**.

Another observation we want to make is that the computation of $digest_T(k, m)$ is done via many ($2M$) logical instructions (AND, OR and XOR) as opposed to a significantly smaller number of MUL and logical instructions in $digest_{WM}(k, m)$. As a consequence, the number of *control* clock cycles (e.g. loop indexing, fetching and writing) involved in $digest_{WM}(k, m)$ is potentially reduced by a factor of $\frac{2M}{2M/b+2M/b} = b/2$ relative to $digest_T(k, m)$.

The word multiplication based method also permits a high level of parallelism in multi-core processors [**?**] thanks to its modular structure, i.e. the digest computation can be easily split into many groups of equal or different sizes of independent word multiplications.

[5] The primary purpose of this new instruction is for computing the Galois Hash, which is the underlying computation of the Galois Counter Mode (GCM) and AES-GCM.

[6] There is no cost of applying Low() and Up() functions to the product of 2 word operands because in a 64/32-bit processor the multiplication result of two 64/32-bit words is typically stored in two 64/32-bit registers which are the upper and lower halves of the product.

7 Statistical tests of word and Toeplitz matrix multiplications

Since if digest keys are derived from a standard pseudorandom number generator (PRNG), which produces ϵ-biased sequences of pseudorandom bits, clearly we should expect good statistical results on Toeplitz based digest construction $digest_T(k, m)$ thanks to its mathematical structure (Theorem **??**).[7] In this section, we concentrate on verifying the quality of word multiplication based digest function $digest_{WM}(k, m)$ with respect to its distribution and collision properties.

In our digest collision tests, we consider N pairs of distinct messages in combination with a large set $\mathcal{K}$ of keys. For each ith pair of messages (m_i, m_i'), x_i denotes the number of distinct keys from set $\mathcal{K}$ of all keys which result in digest collisions. What we want to verify is the following null hypothesis H_0:

> H_0: The observed numbers of collision keys of all N trials $\{x_1, \ldots, x_N\}$ should fall into a binomial distribution: $x_i \sim Binom(\|\mathcal{K}\|, \epsilon)$, where ϵ is the pairwise-collision probability of a digest function and $\|\mathcal{K}\|$ is the cardinality of set $\mathcal{K}$.

The motivation behind this null hypothesis comes from the fact that in an ideal digest function the chance ϵ that a pair of distinct messages digesting to the same value under key k_1 is the same as and independent from under another randomly chosen (different) key k_2. To check the accuracy of this hypothesis, we first carry out the *chi-square test* to compute the corresponding p-value [**?**]:

$$p\text{-value} = \Pr(\mathcal{X}_V^2 > \text{Observed } X^2)$$

Secondly we compare the distribution of $\{x_1 \ldots x_N\}$ against an ideal binomial distribution $Binom(\|\mathcal{K}\|, \epsilon)$ by plotting their quantiles against each other. This is called the *Quantile-Quantile* or Q-Q plot [**?**]. If the two distributions being compared are similar, then the points in the Q-Q plot will approximately lie on the line $y = x$, which is also drawn in each Q-Q plot to illustrate any difference between the two.

To calculate the collision probability of a digest construction, we will look at the maximum x' of the set $\{x_1 \cdots x_N\}$. And suppose that the null hypothesis H_0 is correct then the value of x' should be very near to the maximum median x at which all N trials from the binomial distribution $Binom(\|\mathcal{K}\|, \epsilon)$ are 50% likely to be less than x. This means that

$$(\Pr\{x_i \leq x\})^N = \left(\sum_{t=0}^{x} \binom{\|\mathcal{K}\|}{t} \epsilon^t (1-\epsilon)^{\|\mathcal{K}\|-t}\right)^N \approx \left(\frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} \epsilon^{-\frac{(t-\mu)^2}{2\sigma^2}} dt\right)^N = 1/2$$

Here $\Pr\{x_i \leq x\}$ is the probability that the ith trial (the number of collision keys for the ith pair of messages) is smaller than x, and $\mu = \|\mathcal{K}\|\epsilon$ and $\sigma^2 = \|\mathcal{K}\|\epsilon(1-\epsilon)$. Since all N trials are themselves independent, we need to raise $\Pr\{x_i \leq x\}$ to the power of N. The approximation between the cumulative binomial distribution function and the cumulative normal distribution function is due to the DeMoive-Laplace limit theorem [**?**]. The point x, which is used to check the accuracy of x' (the observed maximum number of collision keys across N pairs of messages), can then be found by using tabulations of the probability mass functions of the normal distribution [**?**]. For example, when $N = 2^{16}$, $\epsilon = 2^{-10}$ and $\|\mathcal{K}\| =$

[7] Statistical test results of Toeplitz based digest functions can be found in additional supporting information section at the end of this paper.

102400, 1024000 and 10240000, then the expected values for observed x's are respectively 142, 1134 and 10424, as given in Tables **??**, **??**, **??** and **??**.

Regarding a digest distribution test, the main difference between this and collision one is that we will fix N distinct pairs of message and digest output, and then for each ith pair (m_i, d_i) we denote y_i the number of keys from set $\mathcal{K}$ satisfying the equation $digest_{WM}(k, m_i) = d_i$. Since for any pair (m_i, d_i), the probability that key k_1 satisfying $digest(k_1, m_i) = d_i$ is the same as and independent from another randomly chosen key k_2, the set $\{y_1 \dots y_N\}$ should also fall into a binomial distribution, i.e. $y_i \sim Binom(\|\mathcal{K}\|, \epsilon)$. As a result, the same methods of analysing collision tests (chi-square test, Q-Q plots and max median) described above apply to distribution tests.

In our tests, we generate 32-bit digest functions, and so it is sufficient to consider 32-bit input messages. The resulting length for digest keys that are pseudorandomly generated in our test is therefore 64 bits. Since digest keys are usually not influenced by an attacker at the point when they are invented in many applications of digest functions [**?**, **?**, **?**, **?**, **?**], we use the Mersenne Twister pseudorandom number generator [**?**] to derive large sets $\mathcal{K}$ of 64-bit keys.

A problem in carrying out collision (and distribution) tests is that it is very expensive when we want to acquire meaningful and reliable results about collision (and distribution) on the whole of a 32-bit digest output.[8] For this reason, we will only consider collisions on groups of 10 adjacent bits of a digest output, which perhaps unexpectedly gives us evidence on any difference in distribution between these group of bits potentially caused by the impact of carry bits in integer multiplication. This also makes the expected probability of a pairwise 10-bit digest collision be $\epsilon = 2^{-10}$ in all of our tests. In particular, we will look at the following three groups of bits: 10LSB, 10MIB and 10MSB which denote the 10 least, the 10 middle and respectively the 10 most significant bits of a 32-bit digest output of $digest_{WM}(k, m)$.

As opposed to digest keys, input messages are public and can be under the control of an attacker, we therefore consider the following four types of input messages:

- Series 1: Pseudorandom messages derived from the Mersenne Twister [**?**];
- Series 2: Sequential (or cluster) messages where the initial message or pair of messages is derived from the Mersenne Twister;
- Series 3: Sparse and cluster messages where the initial pair of messages in collision tests is $(1, 2^8 + 1)$; and
- Series 4: Dense and cluster messages where the initial pair of messages in collision tests is $(2^{32} - 1, 2^{32} - 257)$.

7.1 Analysis of statistical test results

We have implemented our tests relative to three different numbers of keys, i.e. capitals A, B and C denote 102400, 1024000 and 10240000 which are the cardinalities of set $\mathcal{K}$. However, due to lack of space, we will only give p-values of chi-square tests and their Q-Q plots for the

[8] For example, to have around $1024 = 2^{10}$ 32-bit digest collisions for each pair of messages, we need to consider or generate pseudorandomly $2^{32} \times 2^{10} = 2^{42}$ distinct keys. Since there are 2^{16} pairs of messages in our tests, the number of digest computations is $2 \times 2^{16} \times 2^{42} = 2^{59}$ or $2^{59} \times 2^{11} = 2^{70}$ bit operations, i.e. each 32-bit digest computation of a 64-bit key and 32-bit messages requires two 32-bit word multiplication adding up to $2 \times 2^5 \times 2^5 = 2^{11}$ bit operations. This therefore goes beyond the computation capability of our computers, and it will be much more than this for longer digest functions and longer messages.

case of 10240000 keys as presented in Tables **??**, **??**, **??**, **??**, **??** and **??**. We note that to ensure uniformity in our computation of p-values in chi-square tests for collision and distribution, we always use the same set of bins (or intervals) to count the observed occurrences in each interval.[9]

Since a large number of tables reporting the statistical results of our collision and distribution tests considered here do not always make it easy to see their purposes and importance, we summarise the contents of all of these tables in the following figure. Another digest construction termed $digest_{XWM}(k, m)$ will be described at the end of this section.

Table	Digest method	Digest property	$\|\mathcal{K}\|$	N	Tests
??	$digest_{WM}()$	distribution	C	2^{18}	Chi-square and Q-Q plot
??	$digest_{WM}()$	distribution	A,B,C	2^{18}	mean, max (no of keys) and variance
??	$digest_{WM}()$	collision	C	2^{16}	Chi-square and Q-Q plot
??	$digest_{WM}()$	collision	A,B,C	2^{16}	mean, max (no of keys) and variance
??	$digest_{XWM}()$	collision	C	2^{16}	Chi-square and Q-Q plot
??	$digest_{XWM}()$	collision	A,B,C	2^{16}	mean, max (no of keys) and variance
??	$digest_{XWM}()$	distribution	C	2^{18}	Chi-square and Q-Q plot
??	$digest_{XWM}()$	distribution	A,B,C	2^{18}	mean, max (no of keys) and variance
??	$digest_{T}()$	distribution	C	2^{17}	Chi-square and Q-Q plot
??	$digest_{T}()$	distribution	A,B,C	2^{17}	mean, max (no of keys) and variance
??	$digest_{T}()$	collision	C	2^{16}	Chi-square and Q-Q plot
??	$digest_{T}()$	collision	A,B,C	2^{16}	mean, max (no of keys) and variance

It is clear from Tables **??** that the distribution property of $digest_{WM}(k, m)$ approaches optimality (i.e. as in an ideal binomial distribution) because of the right range of the observed p-values and nearly all the points in the Q-Q plots approximately lie on the line $y = x$ in Series 1-4. This observation is strengthened by results reported in Table **??** which indicates that the ratio between max and mean numbers of keys converges to 1 as the cardinality of set $\mathcal{K}$ increases. Please note that to exploit the 10-bit digest output and to discover if any correlation exits among all distinct digest values with respect to the same message, in our distribution tests we always pair each message with each of every possible digest output from 0 to $1023 = 2^{10} - 1$, i.e. in Table **??**, $N = 2^{18}$ pairs are made up from 2^8 distinct messages and 2^{10} digest values. As a result, the mean values of numbers of keys in all distribution tests equal $\|\mathcal{K}\|\epsilon$ exactly as seen in Tables **??**, **??** and **??**. What we are interested in are the distribution properties of $\{y_1 \dots y_N\}$ represented by p-value, Q-Q plot, max and variance.[10]

As regards collision property, Table **??** suggests when input messages are pseudorandomly generated (Series 1-2), the numbers of collision keys in $digest_{WM}(k, m)$ fall into a binomial distribution (i.e. the null hypothesis H_0 is accurate), because both p-values are in the right range and nearly all points in the Q-Q plots lie approximately on the line $y = x$. Moreover, the maximum number of observed collision keys across all N pair of distinct messages, their

[9]The sets $\{x_1, \dots, x_N\}$ and $\{y_1, \dots, y_N\}$ are counted into the following 16 intervals: [0,9650), [9650,9700), [9700,9750), [9750,9800), [9800,9850), [9850,9900), [9900,9950), [9950,10000), [10000,10050),[10050,10100), [10100,10150), [10150,10200), [10200,10250), [10250,10300), [10300,10350), [10350,$+\infty$), and therefore the degree of freedom in all chi-square tests is 15. Here $10000 = 10240000 * 2^{-10} = N\epsilon$ is the expected value of mean of number of keys across N trials in our chi-square tests.

[10]When each message is not paired up with each of every possible digest output, it is highly unlikely that the observed value of mean equals $\|\mathcal{K}\|\epsilon$ exactly. This has been tested and yielded similar statistical results as the above case.

mean and variance values (Series 1-2 of Table **??**) are all near to the expected values in an ideal scenarios. As in distribution tests, the ratio between the maximum number of observed collision keys and Mean converges to 1 with each additional order of magnitude in the size of the set $\mathcal{K}$, i.e. 10-time increase in the number of keys.

We also obtain similar results regarding 10MSB, 10 MIB and 10MSB in both distribution and collision tests of $digest_{WM}(k, m)$ when messages are pseudorandomly generated as seen in Series 1-2 of Tables **??**-**??**, and Series 1-4 of Table **??**-**??**. This suggests that the digest output bits are equivalent in their quality, and thus carry bits in integer multiplication does not introduce much bias into the digest computation regime.

However, when it comes to either sparse or dense (and cluster) messages as seen in Series 3-4 of Table **??**, we detect unsatisfactory behaviours. For example, there are many points in Q-Q plots moving far away (both above and below) from the line $y = x$, and their p-values are extremely small compared to Series 1-2. To confirm the negative impacts of dense and sparse messages, we look at the ratio between Max and Mean numbers of collision keys in Series 3-4 of Table **??**, and realise that it is around 2 and does not seem to improve even when we increase the number of keys. We therefore conclude that the null hypothesis H_0 stated above is not correct in this case.

To fix this weakness, we need to destroy known patterns within input messages. In other words, we want to randomise input messages in an unpredictable, but also deterministic, way prior to them being processed by our digest schemes. One possible and efficient way is to exclusive-or the message with the output of some function $f()$ applying to the key k, resulting in the following improved digest construction:

$$digest_{XWM}(k, m) = digest_{WM}(k, m \oplus f(k))$$

Since k is random and fresh in each protocol session, and so is the modified version of m. In particular, when we try $f(k) = k_1 \oplus k_2$, where k_1 and k_2 are the two (32-bit) halves of key k, then we seem to be able to get around the negative impact caused by very dense or sparse messages (except a bias in the 10MIB relative to 10MSB and 10LSB of a 32-bit digest output) as reported in Series 3-4 of Tables **??**-**??**. Devising good ways to randomise input messages is therefore a subject for future research.

Testing results regarding distribution and collision properties of both $digest_T(k, m)$ and $digest_{XWM}(k, m)$ can be found in the additional supporting information at the end of this paper.

8 Conclusions and future research for short-output functions

There are two main contributions about a new short-output digest function presented in this paper. First, we prove that Toeplitz matrix multiplication coupled with ϵ-biased sequences of (pseudo)random variables can be used to generate digest functions with the properties we require. Secondly, a structural similarity between Toeplitz matrix and the "school book" algorithm for integer long multiplications exists as pointed out in Section **??**. This similarity leads us to introduce more efficient digest computation based on word multiplications which can be calculated fast in any processors. Although there can be bias due to carry bits in an integer multiplication, we have conducted statistical and collision tests to show that no significant difference seems to exist between our two digest constructions.

It is clear that both of our digest constructions ($digest_T(k, m)$ and $digest_{WM}(k, m)$) permit parallel computation in multi-core processors to increase efficiency. This is however

only possible if the key k is either really long, i.e. the same order as message bitlength $K = M + b - 1$, or derived (off-line) from a short number of bits in advance. We note that there is a theoretical bound on the digest (or universal hash) key length which says that the message length can grow exponentially with the key length: $K \geq \log(M/b)$ [**?**]. This suggests that other digest constructions which require shorter key length might exit.

References

[1] `http://www.intel.com/pressroom/archive/releases/20091202comp_sm.htm`

[2] *Simple Pairing White Paper*. See: `www.bluetooth.com/NR/rdonlyres/0A0B3F36-D15F-4470-85A6-F2CCFA26F70F/0/SimplePairing_WP_V10r00.pdf`

[3] `http://software.intel.com/en-us/articles/carry-less-multiplication-and-its-usage-for-computing-the-gcm-mode/`

[4] `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html`

[5] `http://www.sandia.gov/scada/documents/SANDstorm_Submission_2008_10_30.pdf`

[6] M. Abramowitz and I.A. Stegun. *Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables*. ISBN 0-486-61272-4.

[7] N. Alon, O. Goldreich, J. Hastad and R. Peralta. *Simple Constructions of Almost k-wise Independent Random Variables.* In Proceedings of the IEEE Symposium on Foundations of Computer Science, 1990, vol. 2, pp. 544-553.

[8] D.J. Bernstein, H.-C. Chen, M.-S. Chen, C.-M. Cheng, C.-H. Hsiao, T. Lange, Z.-C. Lin, and B.-Y. Yang. *A Billion-mulmods-per-second PC*. Eurocrypt 2009, Rump Session.

[9] J.L. Carter and M.N. Wegman. *Universal Classes of Hash Functions.* Journal of Computer and System Sciences, vol. 18 (1979), pp. 143-154.

[10] I. Damgård. *A Design Principle for Hash Functions*. CRYPTO, LNCS vol. 435 (1989), 416-427.

[11] C. Gehrmann, C. Mitchell and K. Nyberg. *Manual Authentication for Wireless Devices.* RSA Cryptobytes, vol. 7, no. 1, pp. 29-37, 2004.

[12] T. Johansson, G.A. Kabatianskii and B. Smeets. *On the relation between A-Codes and Codes correcting independent errors.* Advances in Cryptology - Eurocrypt 1993, LNCS vol. 765, pp. 1-11.

[13] H. Krawczyk. *New Hash Functions For Message Authentication.* Advances in Cryptology - Eurocrypt 1995, LNCS vol. 921, pp. 301-310.

[14] S. Laur and K. Nyberg. *Efficient Mutual Data Authentication Using Manually Authenticated Strings*. LNCS vol. 4301, pp. 90-107, 2006.

[15] M. Matsumoto. *Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator*. ACM Transactions on Modeling and Computer Simulation 8: 3-1 (1998).

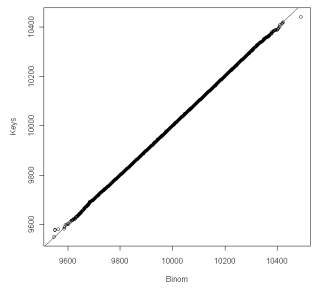
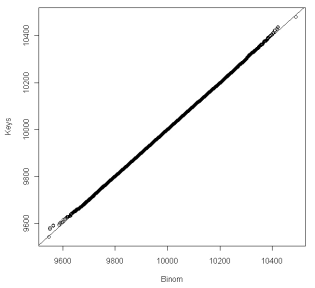
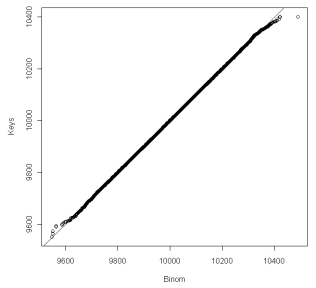
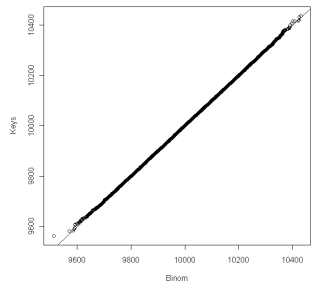
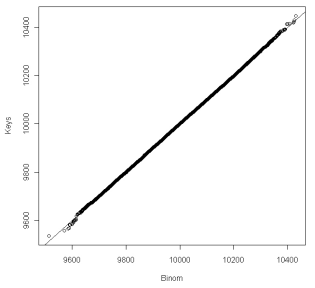
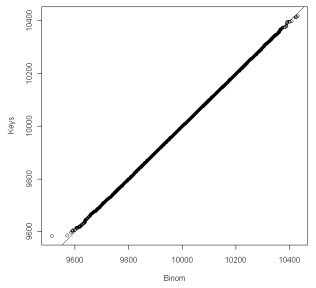
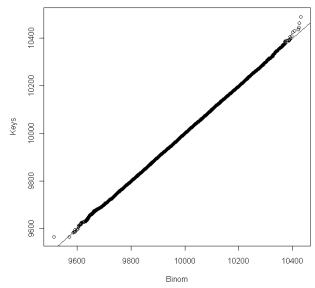
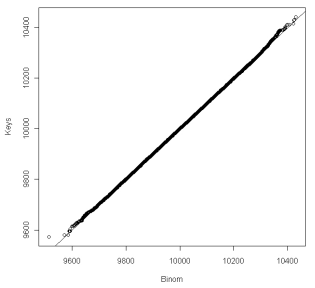
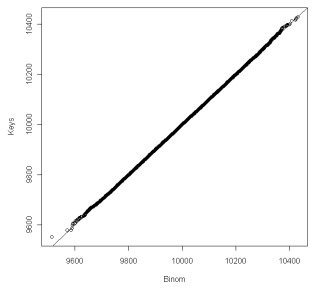
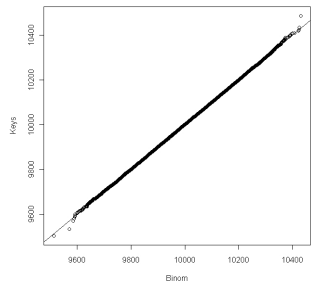
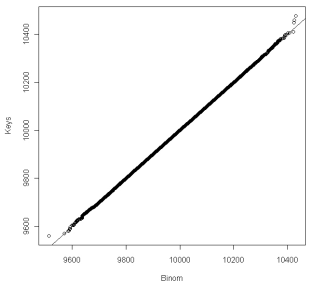
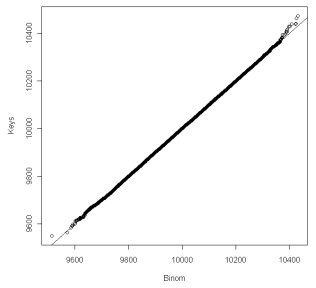
	10LSB	10MIB	10MSB
Pseudorandom 32-bit messages.			
Series 1C	p-value = 0.2316	p-value = 0.1379	p-value = 0.1986
			
Cluster 32-bit messages: the initial message is derived pseudorandomly.			
Series 2C	p-value = 0.5744	p-value = 0.958	p-value = 0.4005
			
Sparse and Cluster 32-bit messages: the initial message is 1.			
Series 3C	p-value = 0.2192	p-value = 0.1688	p-value = 0.2438
			
Dense and Cluster 32-bit messages: the initial message is 2^{32}-1.			
Series 4C	p-value = 0.5379	p-value = 0.905	p-value = 0.6854
			

Table 1: Chi-square and Q-Q plots of distribution test results of 32-bit $digest_{WM}(k, m)$: $N = 2^{18}$ pairs of message and digest output (m, d) over $\|\mathcal{K}\| = 10240000$ keys. In each Q-Q plot, the vertical (Keys) and horizontal (Binom) axes denote quantiles on observed keys and respectively binomial distribution.

Test Series	No of keys $\|\mathcal{K}\|$	Map range	Expected Mean & Max numbers of keys, and Variance	Observed Mean μ'	Observed Max y'	Observed Variance σ'^2
			$\mu = \|\mathcal{K}\|\epsilon$ $\sigma^2 = \|\mathcal{K}\|\epsilon(1-\epsilon)$ $\int_{-\infty}^{y} \epsilon^{-\frac{(t-\mu)^2}{2\sigma^2}} dt = 2^{-1/N}$	$\Sigma_{i=1}^{N} y_i/N$	max of $\{y_1 \dots y_N\}$	$\Sigma_{i=1}^{N} \frac{(y_i-\mu')^2}{N}$
Pseudorandom 32-bit messages.						
1A	102400	10LSB 10MIB 10MSB	100 - 145 - 99.90	100.00 100.00 100.00	148 149 148	100.45 99.86 99.80
1B	1024000	10LSB 10MIB 10MSB	1000 - 1144 - 999.0	1000.00 1000.00 1000.00	1139 1149 1146	1001.43 998.39 1002.97
1C	10240000	10LSB 10MIB 10MSB	10000 - 10445 - 9990	10000.00 10000.00 10000.00	10442 10481 10401	10049.08 9941.39 10027.96
Cluster 32-bit messages: the initial message is derived pseudorandomly.						
2A	102400	10LSB 10MIB 10MSB	100 - 145 - 99.90	100.00 100.00 100.00	148 149 152	99.63 99.94 99.99
2B	1024000	10LSB 10MIB 10MSB	1000 - 1144 - 999.0	1000.00 1000.00 1000.00	1143 1141 1149	998.92 1002.83 1001.46
2C	10240000	10LSB 10MIB 10MSB	10000 - 10445 - 9990	10000.00 10000.00 10000.00	10437 10449 10419	10036.47 9999.81 9944.26
Sparse and Cluster 32-bit messages: the initial message is 1.						
3A	102400	10LSB 10MIB 10MSB	100 - 145 - 99.90	100.00 100.00 100.00	155 149 150	99.38 99.10 99.90
3B	1024000	10LSB 10MIB 10MSB	1000 - 1144 - 999.0	1000.00 1000.00 1000.00	1147 1142 1139	996.47 991.83 987.38
3C	10240000	10LSB 10MIB 10MSB	10000 - 10445 - 9990	10000.00 10000.00 10000.00	10490 10443 10430	10022.99 9914.02 10063.45
Dense and Cluster 32-bit messages: the initial message is $2^{32}-1$.						
4A	102400	10LSB 10MIB 10MSB	100 - 145 - 99.90	100.00 100.00 100.00	155 148 152	99.54 100.31 100.08
4B	1024000	10LSB 10MIB 10MSB	1000 - 1144 - 999.0	1000.00 1000.00 1000.00	1146 1137 1143	1002.80 1001.23 998.89
4C	10240000	10LSB 10MIB 10MSB	10000 - 10445 - 9990	10000.00 10000.00 10000.00	10486 10477 10473	9998.82 9986.94 10032.88

Table 2: Comparing the mean, max (no of keys) and variance of distribution test results of 32-bit $digest_{WM}(k, m)$ against their expected values in an ideal binomial distribution: $N = 2^{18}$ distinct pairs of message and digest output (m, d). The set $\{y_1 \dots y_N\}$ denotes the number of keys observed and counted for all N pairs (m, d).

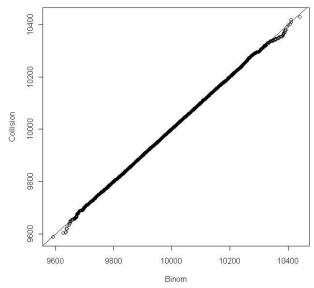
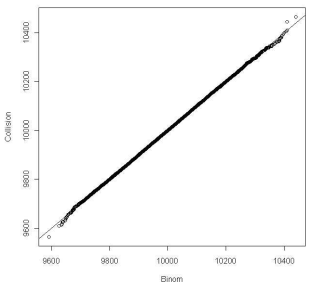
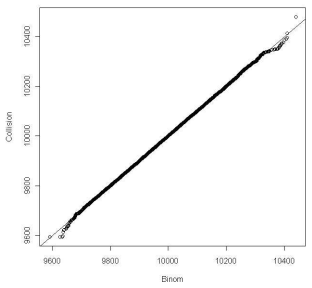
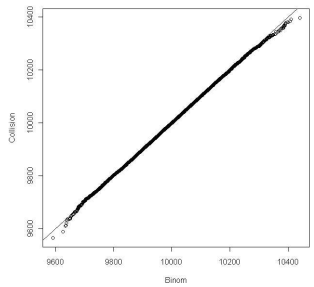
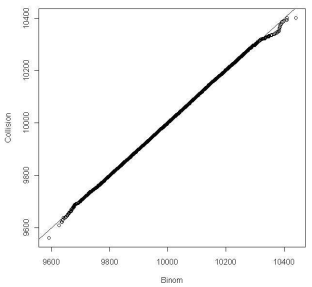
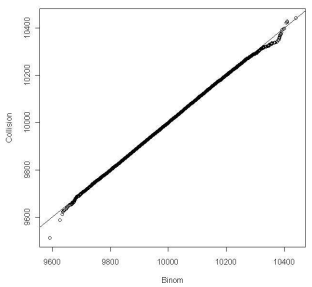
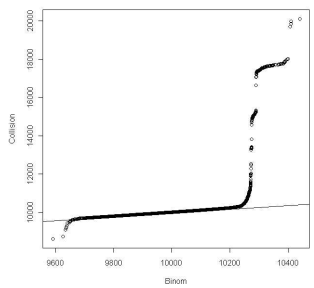
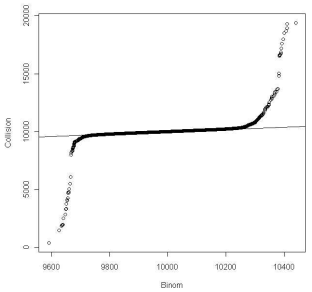
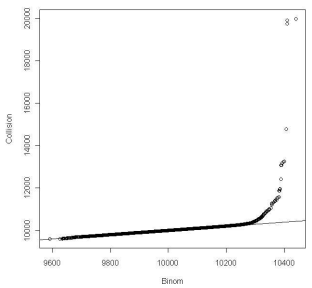
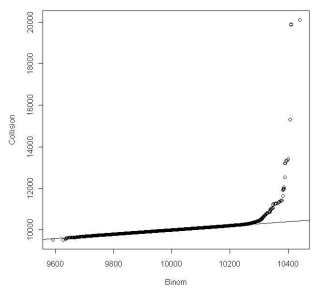
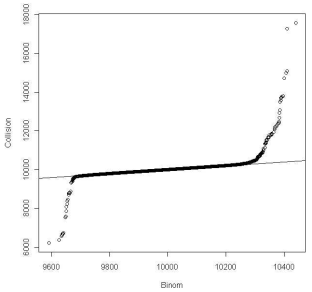
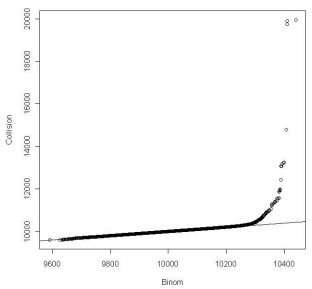
	10LSB	10MIB	10MSB
Pseudorandom 32-bit messages.			
Series 1C	p-value = 0.6868	p-value = 0.4816	p-value = 0.4816
			
Cluster 32-bit messages: the initial pair of messages is derived pseudorandomly.			
Series 2C	p-value = 0.6366	p-value = 0.5683	p-value = 0.3670
			
Sparse and Cluster 32-bit messages: the initial pair of messages is $(1,2^8+1)$.			
Series 3C	p-value = 2.2e-16	p-value = 2.2e-16	p-value = 2.2e-16
			
Dense and Cluster 32-bit messages: the initial pair of messages is $(2^{32}-1, 2^{32}-257)$.			
Series 4C	p-value = 2.2e-16	p-value = 2.2e-16	p-value = 2.2e-16
			

Table 3: Chi-square and Q-Q plots of collision test results of 32-bit $digest_{WM}(k,m)$: $N = 2^{16}$ pairs of 32-bit messages over $\|\mathcal{K}\| = 10240000$ keys. In each Q-Q plot, the vertical (Collision) and horizontal (Binom) axes denote quantiles on observed collision keys and respectively binomial distribution.

Test Series	No of keys $\|\mathcal{K}\|$	Map range	Expected Mean & Max numbers of collision keys, and Variance	Observed Mean μ'	Observed Max x'	Observed Variance σ'^2
			$\mu = \|\mathcal{K}\|\epsilon$ $\sigma^2 = \|\mathcal{K}\|\epsilon(1-\epsilon)$ $\int_{-\infty}^{x} \epsilon^{-\frac{(t-\mu)^2}{2\sigma^2}} dt = 2^{-1/N}$	$\Sigma_{i=1}^{N} x_i/N$	max of $\{x_1 \ldots x_N\}$	$\Sigma_{i=1}^{N} \frac{(x_i-\mu')^2}{N}$
Pseudorandom 32-bit messages.						
1A	102400	10LSB 10MIB 10MSB	100 - 142 - 99.90	100.10 99.61 99.99	156 162 140	99.5 98.1 99.8
1B	1024000	10LSB 10MIB 10MSB	1000 - 1134 - 999.0	999.91 999.98 999.97	1153 1128 1134	993.9 994.3 1000.5
1C	10240000	10LSB 10MIB 10MSB	10000 - 10424 - 9990	10000.73 10000.36 9999.99	10412 10433 10404	10039.8 10010.3 10102.2
Cluster 32-bit messages: the initial pair of messages is derived pseudorandomly.						
2A	102400	10LSB 10MIB 10MSB	100 - 142 - 99.90	100.05 99.97 100.04	153 143 145	99.0 100.4 99.1
2B	1024000	10LSB 10MIB 10MSB	1000 - 1134 - 999.0	1000.07 1000.03 999.68	1135 1133 1146	1004.7 1000.6 1004.4
2C	10240000	10LSB 10MIB 10MSB	10000 - 10424 - 9990	9999.74 10000.10 10000.41	10406 10431 10415	10037.5 10026.4 9998.5
Sparse and Cluster 32-bit messages: the initial pair of messages is $(1,2^8+1)$.						
3A	102400	10LSB 10MIB 10MSB	100 - 142 - 99.90	100.32 100.02 100.12	214 195 224	114.2 101.8 99.99
3B	1024000	10LSB 10MIB 10MSB	1000 - 1134 - 999.0	1002.80 1000.27 1000.01	2059 1976 1998	2434.8 1355.1 1061.6
3C	10240000	10LSB 10MIB 10MSB	10000 - 10424 - 9990	10027.32 10003.16 10005.45	20123 19348 20047	155437.4 44738.7 17346.7
Dense and Cluster 32-bit messages: the initial pair of messages is $(2^{32}-1, 2^{32}-257)$.						
4A	102400	10LSB 10MIB 10MSB	100 - 142 - 99.90	100.06 100.02 100.18	202 163 224	99.8 100.3 100.2
4B	1024000	10LSB 10MIB 10MSB	1000 - 1134 - 999.0	1000.29 1000.45 999.97	2059 1692 2003	1069.7 1077.6 1062.7
4C	10240000	10LSB 10MIB 10MSB	10000 - 10424 - 9990	10003.03 10003.72 10005.41	19905 17478 20090	17230.5 19184.4 17413.0

Table 4: Comparing the mean, max (no of keys) and variance of collision test results of 32-bit $digest_{WM}(k,m)$ against their expected values in an ideal binomial distribution: $N = 2^{16}$ pairs of 32-bit messages. The set $\{x_1 \ldots x_N\}$ denotes the number of collision keys observed and counted for all N trials.

[16] R.C. Merkle. *A digital signature based on a conventional encryption function.* Crypto 1987, LNCS vol. 293, pp. 369 - 378.

[17] L.H. Nguyen and A.W. Roscoe. *Authenticating ad hoc networks by comparison of short digests.* Information and Computation, vol. 206 (2008), pp. 250-271. Proceedings of Workshop on Foundation of Computer Security and Automated Reasoning Protocol Security Analysis, pp. 9-31, 2006.

[18] L.H. Nguyen and A.W. Roscoe. *Separating two roles of hashing in one-way message authentication.* Proceedings of FCS-ARSPA-WITS 2008, pp. 195-210.

[19] S. Pasini and S. Vaudenay. *SAS-based Authenticated Key Agreement.* Public Key Cryptography - PKC 2006: The 9th international workshop on theory and practice in public key cryptography, LNCS vol. 3958, pp. 395-409.

[20] J.A. Rice *Mathematical Statistics and data Analysis.* ISBN 0-534-20934-3

[21] R.L. Rivest. *The MD6 Hash Function.* Crypto 2008.

[22] S. Ross. *A First Course in Probability.* ISBN 0-13-033851-6.

[23] D.R. Stinson. *Universal Hashing and Authentication Codes.* Advances in Cryptology - Crypto 1991, LNCS vol. 576, pp. 74-85, 1992.

[24] M.N. Wegman and J.L. Carter. *New Hash Functions and Their Use in Authentication and Set Equality.* Journal of Computer and System Sciences, vol. 22, pp. 265-279, 1981.

A Additional supporting information about statistical test results of Toeplitz based and improved word multiplication based digest functions

Please see Tables **??-??** for testing results of $digest_T(k, m)$, and Tables **??-??** for testing results of $digest_{XWM}(k, m)$.

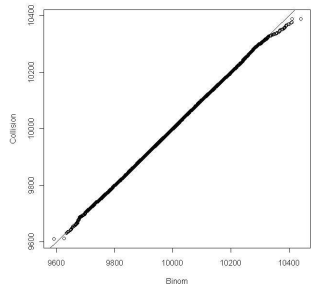
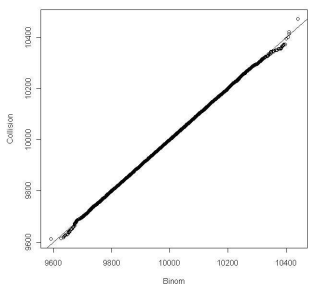
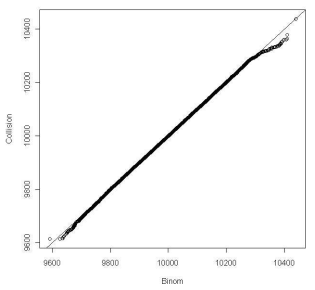
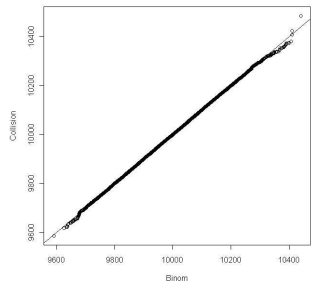
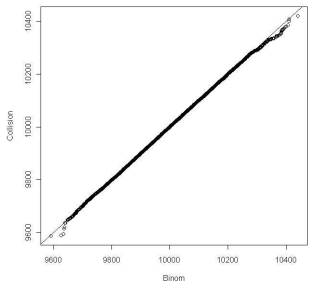
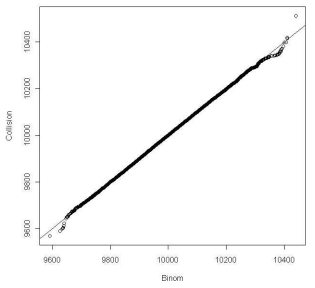
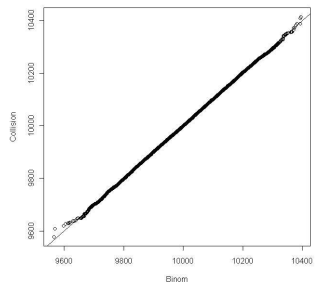
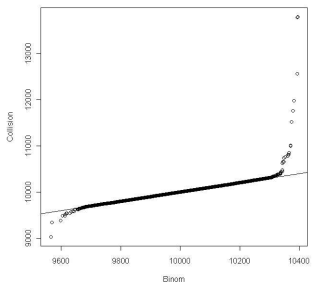
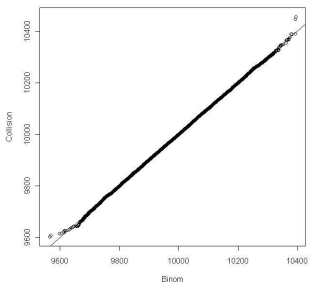
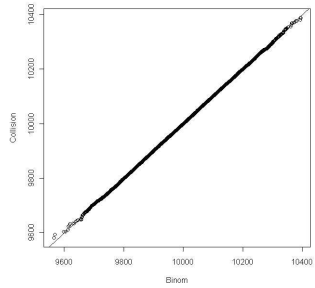
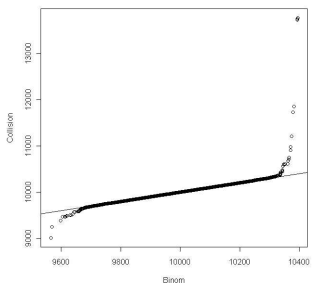
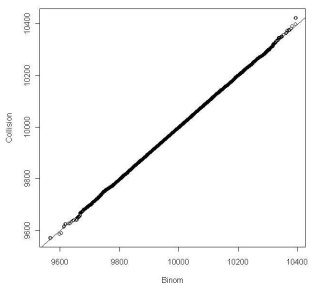
	10LSB	10MIB	10MSB
Pseudorandom 32-bit messages.			
Series 1C	p-value = 0.1139	p-value = 0.9219	p-value = 0.090
			
Cluster 32-bit messages: the initial pair of messages is derived pseudorandomly.			
Series 2C	p-value = 0.9015	p-value = 0.2727	p-value = 0.5111
			
Sparse and Cluster 32-bit messages: the initial pair of messages is $(1,2^8+1)$.			
Series 3C	p-value = 0.0730	p-value = 4.155e-10	p-value = 0.4952
			
Dense and Cluster 32-bit messages: the initial pair of messages is $(2^{32}-1, 2^{32}-257)$.			
Series 4C	p-value = 0.6578	p-value = 6.616e-12	p-value = 0.4497
			

Table 5: Chi-square and Q-Q plots of collision test results of 32-bit $digest_{XWM}(k,m)$ (input messages are XORed with two 32-bit halves of keys prior to multiplication): $N = 2^{16}$ pairs of 32-bit messages over $\|\mathcal{K}\| = 10240000$ distint keys. In each Q-Q plot, the vertical (Collision) and horizontal (Binom) axes denote quantiles on observed collision keys and respectively binomial distribution.

Test Series	No of keys $\|\mathcal{K}\|$	Map range	Expected Mean & Max numbers of collision keys, and Variance	Observed Mean μ'	Observed Max x'	Observed Variance σ'^2
			$\mu = \|\mathcal{K}\|\epsilon$ $\sigma^2 = \|\mathcal{K}\|\epsilon(1-\epsilon)$ $\int_{-\infty}^{x} \epsilon^{-\frac{(t-\mu)^2}{2\sigma^2}} dt = 2^{-1/N}$	$\Sigma_{i=1}^{N} x_i/N$	max of $\{x_1 \ldots x_N\}$	$\Sigma_{i=1}^{N} \frac{(x_i-\mu')^2}{N}$
Pseudorandom 32-bit messages.						
1A	102400	10LSB 10MIB 10MSB	100 - 142 - 99.90	99.96 100.00 100.10	141 147 143	99.5 98.1 100.1
1B	1024000	10LSB 10MIB 10MSB	1000 - 1134 - 999.0	1000.01 999.89 1000.05	1136 1148 1143	997.2 1010.0 996.6
1C	10240000	10LSB 10MIB 10MSB	10000 - 10424 - 9990	9999.82 10000.52 9999.73	10389 10474 10439	10007.3 10033.5 9996.1
Cluster 32-bit messages: the initial pair of messages is derived pseudorandomly.						
2A	102400	10LSB 10MIB 10MSB	100 - 142 - 99.90	99.98 100.01 100.02	150 144 148	100.1 99.7 99.1
2B	1024000	10LSB 10MIB 10MSB	1000 - 1134 - 999.0	999.91 999.98 1000.00	1159 1132 1142	997.6 999.6 994.5
2C	10240000	10LSB 10MIB 10MSB	10000 - 10424 - 9990	9999.89 10000.17 9999.82	10485 10417 10430	9937.5 10037.8 10073.5
Sparse and Cluster 32-bit messages: the initial pair of messages is $(1,2^8+1)$.						
3A	102400	10LSB 10MIB 10MSB	100 - 142 - 99.90	100.01 100.02 100.14	147 143 147	99.6 98.4 99.2
3B	1024000	10LSB 10MIB 10MSB	1000 - 1134 - 999.0	999.84 1000.04 999.79	1129 1359 1133	1002.1 1010.3 1002.1
3C	10240000	10LSB 10MIB 10MSB	10000 - 10424 - 9990	9999.58 10000.69 10002.48	10427 13734 10425	10058.8 10834.1 9905.4
Dense and Cluster 32-bit messages: the initial pair of messages is $(2^{32}-1, 2^{32}-257)$.						
4A	102400	10LSB 10MIB 10MSB	100 - 142 - 99.90	99.99 99.95 100.13	145 144 145	98.7 98.8 99.4
4B	1024000	10LSB 10MIB 10MSB	1000 - 1134 - 999.0	999.70 1000.01 999.95	1142 1374 1143	990.1 1004.7 994.4
4C	10240000	10LSB 10MIB 10MSB	10000 - 10424 - 9990	9999.48 9999.25 10001.97	10398 13707 10503	10044.7 11166.1 9926.0

Table 6: Comparing the mean, max (no of keys) and variance of collision test results of 32-bit $digest_{XWM}(k,m)$ (input messages are XORed with two 32-bit halves of the key prior to multiplication) against their expected values in an ideal binomial distribution: $N = 2^{16}$ pairs of 32-bit messages. The set $\{x_1 \ldots x_N\}$ denotes the number of collision keys observed and counted for all N trials.

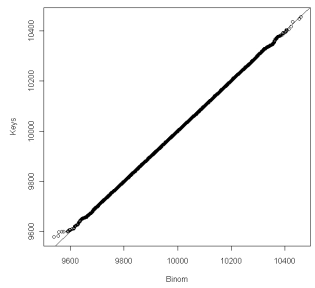
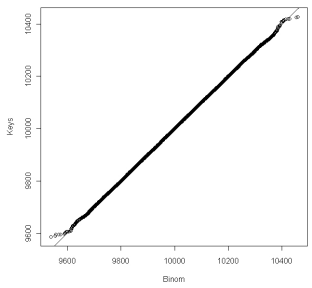
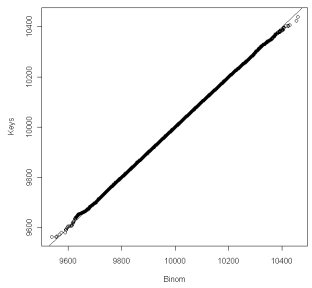
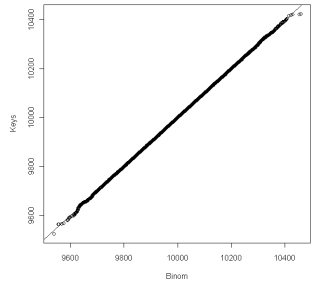
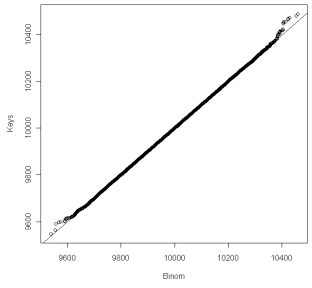
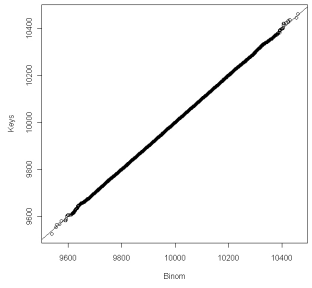
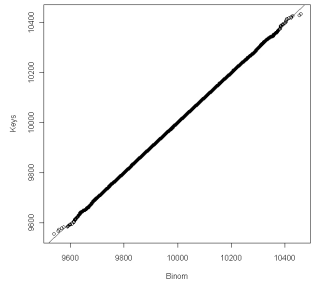
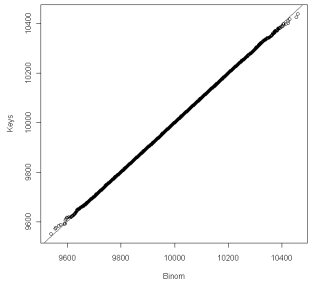
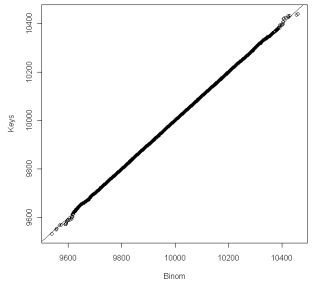
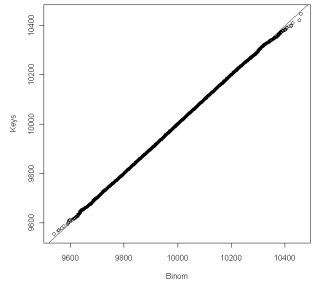
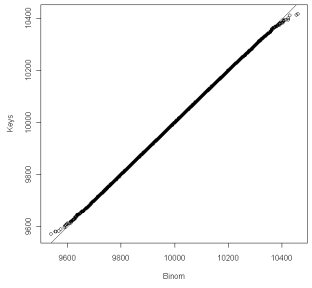
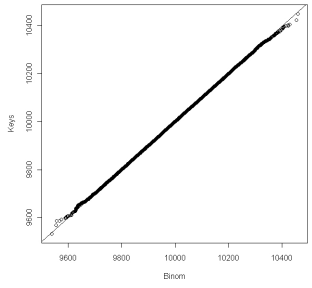
	10LSB	10MIB	10MSB
Pseudorandom 32-bit messages.			
Series 1C	p-value = 0.1577	p-value = 0.3704	p-value = 0.1214
			
Cluster 32-bit messages: the initial message is derived pseudorandomly.			
Series 2C	p-value = 0.7553	p-value = 0.8318	p-value = 0.934
			
Sparse and Cluster 32-bit messages: the initial message is 1.			
Series 3C	p-value = 0.1064	p-value = 0.9174	p-value = 0.6268
			
Dense and Cluster 32-bit messages: the initial message is $2^{32}-1$.			
Series 4C	p-value = 0.0003124	p-value = 0.5064	p-value = 0.6865
			

Table 7: Chi-square and Q-Q plots of distribution test results of 32-bit $digest_{XWM}(k, m)$ (input messages are XORed with two 32-bit halves of keys prior to multiplication): $N = 2^{18}$ pairs of 32-bit messages over $\|\mathcal{K}\| = 10240000$ distint keys. In each Q-Q plot, the vertical and horizontal axes denote quantiles on observed (collision) keys and respectively binomial distribution.

Test Series	No of keys $\|\mathcal{K}\|$	Map range	Expected Mean & Max numbers of keys, and Variance	Observed Mean μ'	Observed Max y'	Observed Variance σ'^2
			$\mu = \|\mathcal{K}\|\epsilon$ $\sigma^2 = \|\mathcal{K}\|\epsilon(1-\epsilon)$ $\int_{-\infty}^{y} \epsilon^{-\frac{(t-\mu)^2}{2\sigma^2}} dt = 2^{-1/N}$	$\Sigma_{i=1}^{N} y_i/N$	max of $\{y_1 \dots y_N\}$	$\Sigma_{i=1}^{N} \frac{(y_i-\mu')^2}{N}$
Pseudorandom 32-bit messages.						
1A	102400	10LSB 10MIB 10MSB	100 - 145 - 99.90	100.00 100.00 100.00	151 151 158	100.34 99.76 99.71
1B	1024000	10LSB 10MIB 10MSB	1000 - 1144 - 999.0	1000.00 1000.00 1000.00	1146 1160 1165	995.60 999.70 1002.67
1C	10240000	10LSB 10MIB 10MSB	10000 - 10445 - 9990	10000.00 10000.00 10000.00	10456 10428 10439	10031.05 9968.10 9907.97
Cluster 32-bit messages: the initial message is derived pseudorandomly.						
2A	102400	10LSB 10MIB 10MSB	100 - 145 - 99.90	100.00 100.00 100.00	146 149 148	99.80 99.67 100.06
2B	1024000	10LSB 10MIB 10MSB	1000 - 1144 - 999.0	1000.00 1000.00 1000.00	1142 1141 1150	997.07 997.58 1000.52
2C	10240000	10LSB 10MIB 10MSB	10000 - 10445 - 9990	10000.00 10000.00 10000.00	10425 10490 10463	10038.38 9983.42 10009.36
Sparse and Cluster 32-bit messages: the initial message is 1.						
3A	102400	10LSB 10MIB 10MSB	100 - 145 - 99.90	100.00 100.00 100.00	149 151 153	99.67 100.12 99.94
3B	1024000	10LSB 10MIB 10MSB	1000 - 1144 - 999.0	1000.00 1000.00 1000.00	1149 1150 1137	996.76 999.42 997.70
3C	10240000	10LSB 10MIB 10MSB	10000 - 10445 - 9990	10000.00 10000.00 10000.00	10436 10440 10440	9989.51 9960.43 9989.53
Dense and Cluster 32-bit messages: the initial message is $2^{32}-1$.						
4A	102400	10LSB 10MIB 10MSB	100 - 145 - 99.90	100.00 100.00 100.00	153 147 147	100.15 99.79 99.78
4B	1024000	10LSB 10MIB 10MSB	1000 - 1144 - 999.0	1000.00 1000.00 1000.00	1144 1145 1150	1001.88 1002.80 995.98
4C	10240000	10LSB 10MIB 10MSB	10000 - 10445 - 9990	10000.00 10000.00 10000.00	10447 10418 10450	9982.14 9987.62 10011.10

Table 8: Comparing the mean, max (no of keys) and variance of distribution test results of 32-bit $digest_{XWM}(k, m)$ against their expected values in an ideal binomial distribution: $N = 2^{18}$ distinct pairs of message and digest output (m, d). The set $\{y_1 \dots y_N\}$ denotes the number of keys observed and counted for all N pairs (m, d).

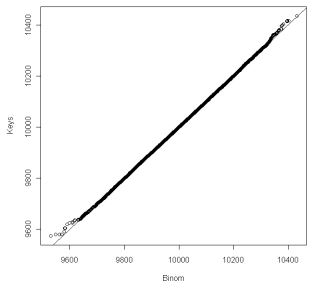
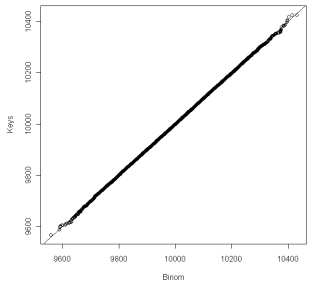
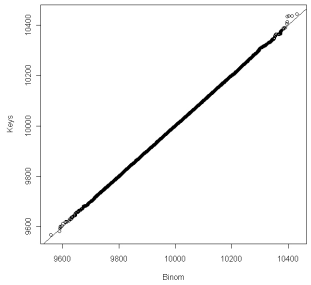
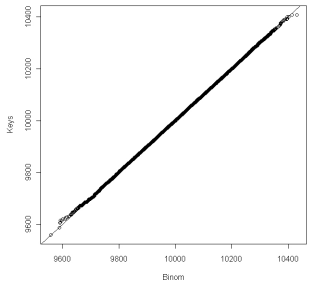
Pseudorandom 32-bit messages	
Series 1C	p-value = 0.1684 
Cluster 32-bit messages: the initial message is derived pseudorandomly.	
Series 2C	p-value = 0.283 
Sparse and Cluster 32-bit messages: the initial message is 1.	
Series 3C	p-value = 0.627 
Dense and Cluster 32-bit messages: the initial message is $2^{32} - 1$.	
Series 4C	p-value = 0.227 

Table 9: Chi-square and Q-Q plots of distribution test results of 32-bit $digest_T(k, m)$: $N = 2^{17}$ pairs of 32-bit messages over $\|\mathcal{K}\| = 10240000$ keys. In each Q-Q plot, the vertical and horizontal axes denote quantiles on observed keys and respectively binomial distribution.

Test Series	No of 64-bit random keys $\|\mathcal{K}\|$	Expected Mean & Max numbers of keys, and Variance	Observed Mean μ'	Observed Max x'	Observed Variance σ'^2
		$\mu = \|\mathcal{K}\|\epsilon$ $\sigma^2 = \|\mathcal{K}\|\epsilon(1-\epsilon)$ $\int_{-\infty}^{x} \epsilon^{-\frac{(t-\mu)^2}{2\sigma^2}} dt = 2^{-1/N}$	$\Sigma_{i=1}^{N} y_i/N$	maximum of $\{y_1 \dots y_N\}$	$\Sigma_{i=1}^{N} \frac{(y_i-\mu')^2}{N}$
Pseudorandom 32-bit messages.					
1A	102400	100 - 144 - 99.90	100.00	145	100.19
1B	1024000	1000 - 1139 - 999.0	1000.00	1147	1002.38
1C	10240000	10000 - 10439 - 9990	10000.00	10436	9997.77
Cluster 32-bit messages: the initial message is derived pseudorandomly.					
2A	102400	100 - 144 - 99.90	100.00	148	99.95
2B	1024000	1000 - 1139 - 999.0	1000.00	1152	1003.82
2C	10240000	10000 - 10439 - 9990	10000.00	10427	9920.94
Sparse and Cluster 32-bit messages: the initial message is 1.					
3A	102400	100 - 144 - 99.90	100.00	145	99.53
3B	1024000	1000 - 1139 - 999.0	1000.00	1130	982.61
3C	10240000	10000 - 10439 - 9990	10000.00	10445	9949.84
Dense and Cluster 32-bit messages: the initial message is $2^{32}-1$.					
4A	102400	100 - 144 - 99.90	100.00	143	100.16
4B	1024000	1000 - 1139 - 999.0	1000.00	1143	998.97
4C	10240000	10000 - 10439 - 9990	10000.00	10409	9910.73

Table 10: Comparing the mean, max (no of keys) and variance of distribution test results of 32-bit $digest_T(k, m)$ (Toeplitz method) against their expected values in an ideal binomial distribution: $N = 2^{17}$ pairs of 32-bit messages. The set $\{y_1 \dots y_N\}$ denotes the number of collision keys observed and counted for all N trials.

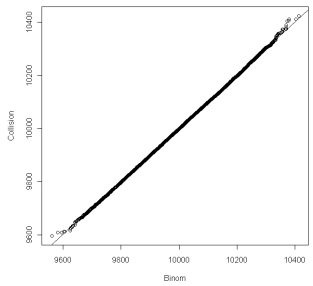
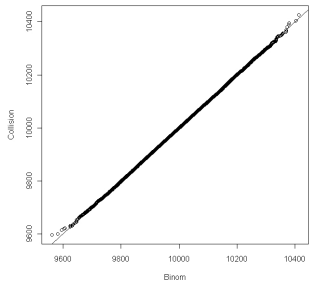
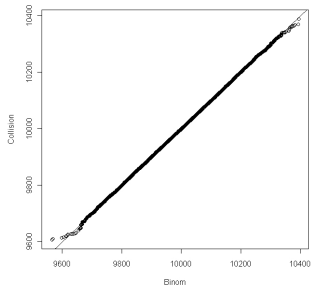
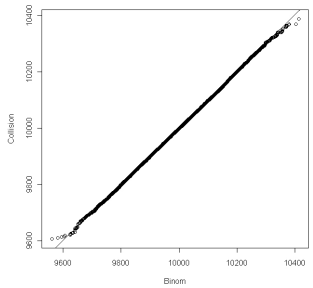
Pseudorandom 32-bit messages	
Series 1C	p-value = 0.4752 
Cluster 32-bit messages: the initial pair of messages is derived pseudorandomly.	
Series 2C	p-value = 0.2469 
Sparse and Cluster 32-bit messages: the initial pair of messages is $(1,2^8+1)$.	
Series 3C	p-value = 0.05804 
Dense and Cluster 32-bit messages: the initial pair of messages is $(2^{32}-1, 2^{32}-257)$.	
Series 4C	p-value = 0.06694 

Table 11: Chi-square and Q-Q plots of collision test results of 32-bit $digest_T(k,m)$: $N = 2^{16}$ pairs of 32-bit messages over $\|\mathcal{K}\| = 10240000$ keys. In each Q-Q plot, the vertical and horizontal axes denote quantiles on observed keys and respectively binomial distribution.

Test Series	No of 64-bit random keys $\|\mathcal{K}\|$	Expected Mean & Max number of collision keys and Variance	Observed Mean μ'	Observed Max x'	Observed Variance σ'^2
		$\mu = \|\mathcal{K}\|\epsilon$ $\sigma^2 = \|\mathcal{K}\|\epsilon(1-\epsilon)$ $\int_{-\infty}^{x} \epsilon^{-\frac{(t-\mu)^2}{2\sigma^2}} dt = 2^{-1/N}$	$\Sigma_{i=1}^{N} x_i/N$	maximum of $\{x_1 \ldots x_N\}$	$\Sigma_{i=1}^{N} \frac{(x_i-\mu')^2}{N}$
Pseudorandom 32-bit messages.					
1A	102400	100 - 142 - 99.90	99.99	146	99.1
1B	1024000	1000 - 1134 - 999.0	1000.12	1153	997.6
1C	10240000	10000 - 10424 - 9990	10000.69	10425	9958.3
Cluster 32-bit messages: the initial pair of messages is derived pseudorandomly.					
2A	102400	100 - 142 - 99.90	100.04	146	99.4
2B	1024000	1000 - 1134 - 999.0	1000.09	1144	993.3
2C	10240000	10000 - 10424 - 9990	10000.31	10409	10016.5
Sparse and Cluster 32-bit messages: the initial pair of messages is $(1,2^8+1)$.					
3A	102400	100 - 142 - 99.90	100.20	142	99.6
3B	1024000	1000 - 1134 - 999.0	999.85	1137	1003.7
3C	10240000	10000 - 10424 - 9990	9998.26	10444	10005.6
Dense and Cluster 32-bit messages: the initial pair of messages is $(2^{32}-1, 2^{32}-257)$.					
4A	102400	100 - 142 - 99.90	100.06	142	99.7
4B	1024000	1000 - 1134 - 999.0	1000.16	1127	1005.6
4C	10240000	10000 - 10424 - 9990	9999.42	10461	9940.6

Table 12: Comparing the mean, max (no of keys) and variance of collision test results of 32-bit $digest_T(k,m)$ (Toeplitz method) against their expected values in an ideal binomial distribution: $N = 2^{16}$ pairs of 32-bit messages. The set $\{x_1 \ldots x_N\}$ denotes the number of collision keys observed and counted for all N trials.