Secure and Efficient Random Functions with Variable-Length Output

Yan Zhu[a], Di Ma[b], Changjun Hu[a], Gail-Joon Ahn[c], Hongxin Hu[d]

[a] *University of Science and Technology Beijing, Beijing, China 100083*
[b] *University of Michigan-Dearborn, Dearborn, Michigan 48128*
[c] *Arizona State University, Tempe, Arizona, 85287*
[d] *Delaware State University, Dover, Delaware, 19901*

Abstract

Many random functions, like Hash, MAC, PRG, have been used in various network applications for different security choices, however they either are fast but insecure, or are cryptographic secure but slow. To integrate them together, in this paper we present a new family of square random functions, including SqHash, SqMAC and SqPRG, based on a specially truncated function (MSB or LSB), as well as circular convolution with carry bits. Provable security is provided by privacy property in hidden number problem and Hard-core unpredication of one-way function. The experiment results show that these schemes have better performance under different input and output lengths. We perform four types of statistical tests for randomness. The experiments indicate that our construction has good average-case randomness than SHA-2 and original Square algorithm.

Keywords: Algorithm, Randomness, Hash Function, Proformance

1. Introduction

Cryptographic primitives, such as Hash function, message authentication code (MAC), and pseudorandom generator (PRG), have been widely as the cornerstones of various cryptosystems for key/random number generation, message authentication, key exchange, and so on. A common feature among these functions is that their outputs are pseudo-random. So these functions are collectively called *random function*. In cryptography, a random function is essentially a one-way function, which is easy to compute on any input, but hard to invert given an output value.

A (cryptographic) hash function is one of the most basic forms of random function. It has security properties such as preimage resistance, second-preimage resistance, and collision resistance. Based on the underlying building block, existing hash functions can be divided into two categories with or without **provable security**. Hash functions in the first category have provable security. They are usually built upon some computationally hard problems and their security can be proved with rigorous security proofs. These hash functions are commonly called as *provable secure cryptographic hash functions*. However, these functions usually run too slow, especially compared with their peers in the second category.

In contrary, hash functions in the second category do not have provable security but have better performance. They are normally designed on a more-or-less Ad-Hoc basis, where bits of the input message are strategically mixed and feedbacked over multiple rounds to produce the hash output. They are believed to be secure, satisfying the security properties required by a hash function. However, no formal proof can be given. Although lacking provable security, hash functions in the second category have been largely used in the real world due to their high performance. Standard hash functions such as MD5, SHA-1 and SHA-2 belong to this category. The use of cryptographic primitives with no provable security puts our systems at risk since we do not know when and how they can be broken. Some of hash functions, including MD5 and SHA-1, are already broken or found with much less level of security Wang and Yu (2005). However, due to the importance of random functions in building secure systems and their ubiquitous use on various platforms such as phone or PDA, there is a need to have secure random functions with formal security assurance.

A new cryptographic technique, *hidden number problem* (HNP) on lattice theory, provides us with a powerful tool for constructing a desired family of random functions. For instance, it is well-known that the Square (Blum Blum Shub) hash algorithm, is provably secure, based on the difficulty of the quadratic residue problem (QRP). But only $(\log \log N)$ lower-order bits Blum et al. (1986) is considered as random in the output of QRP modulo a composite N, e.g., for a 1024-bit N, we can get a 10-bit random output. However, according to the new research result on HNP, we known that $(\log N)/3$ bits Boneh and Venkatesan (1997); Kiltz (2001) still remain random. This means that we can get about 341-bit random output. Therefore, HNP should be an effective optimization technique to develop an efficient, provably secure, variable-length random function. Variable-length output of

functions bring us great convenience in network applications.

In this paper, we first analyze the security and performance features of the existing Square Hash function. We show several shortcomings of it. In response to these shortcomings, we present a new hash scheme (called SqHash) based on a specially truncated function (most significant bits, MSB). We further improve the performance of SqHash by using "circular convolution" which makes variable-length output possible. Similarly, we present a new MAC scheme (called SqMAC) and a new PRG scheme (called SqPRG). We also prove that the security of these constructions based on privacy property in hidden number problem and Hard-core unpredication of one-way function. The experiment results show that these schemes have better performance under different input and output lengths. We also perform 4 types of statistical tests for randomness. The experiments indicate that our construction has good average-case randomness than SHA-2 and original Square algorithm. The proposed schemes as well as their respective performance and security parameters are summarized in Table 1.

Table 1: Summary of our proposed schemes.

Name	Item	Description
SqHash	Equation	$MSB_k((m\|\|IV) *'_c (m\|\|IV))$
	Performance	$O(n * k)$
	Security	Preimage resistance and Collision resistance
SqMAC	Equation	$MSB_k((m+K) *'_c (m+K))$
	Performance	$O(n * k)$
	Security	Secret-key privacy on Hidden Number Problem
SqPRG	Equation	$\widetilde{LSB}^a_{2l}((x+i+l) *'_c (x+i+l))$
	Performance	$O(n * 2l)$
	Security	Pseudorandom on Hard-core unpredictability

The remainder of this paper is organized as follows. In Section 2 and 3, we review some preliminary background. In Section 4, 5 and 6, we present the new Hash, MAC and PRG constructions. In Section 7, 8 and 9, we analyze the security and performance features of our schemes and the improved schemes. Section 10 concludes the paper.

2. Related Work

Hash function is one of the most basic forms of random function. It is related to (and often confused with) checksum, fingerprint, randomization function, and so on. Generally, hash functions can be divided into two main categories: non-cryptographic hash functions and cryptographic hash functions. The former neglects cryptographic security, so it is not cryptographically strong, but it offers these benefits: 1) it is extremely simple; 2) it is performed on bitwise and bit shift operations; and 3) it executes quickly on resource-limited processors. In Table 2 we show some non-cryptographic hash functions, such as, Spooky, FNV, Lookup3, and Murmur Jenkins (2009); Fowler et al. (1988); Appleby (2011). However, this high performance makes it more feasible for a computer to find hash values (and thus collisions) by brute-force.

Table 2: Hash Function Collection

Type	Hash Fct.	Invented by	Ref
Non-Crypt. Hash	Lookup3	Bob Jenkins	Jenkins (2009)
	Spooky	Bob Jenkins	Jenkins (2009)
	FNV	Fowler, Noll, Vo	Fowler et al. (1988)
	Murmur	Austin Appleby	Appleby (2011)
Crypt. Hash	MD5	Ronald Rivest	Sasaki and Aoki (2009)
	SHA	NSA	Sasaki and Aoki (2009)
	FSB	Augot, Finiasz, etc	Srinathan et al. (2007)
MAC	CBC-MAC	FIPS	Bellare et al. (2000)
	HMAC	Bellare, Canetti, etc	Bellare et al. (1996)
PRG	LCG*		
	LFSR+		
	BBS	Blum, Shub, etc	Blum et al. (1986)

* Linear congruential generator. +Linear feedback shift register.

A cryptographic hash function is a cryptographic algorithm which is able to resist all types of attack. As a minimum, it must have the following properties: preimage resistance, second-preimage resistance, and collision resistance. Some existing schemes are designed on a mathematical problem and thus their security follows from rigorous mathematical proofs for properties. Such a function is called provably secure cryptographic hash functions, such as Square Blum et al. (1986). Another existing schemes are not based

on mathematical problems but on an Ad-Hoc basis, where the bits of the message are mixed to produce the hash. They are then believed to be hard to break, but no such formal proof is given. Almost all widely-spread hash functions fall in this category. For example, we shown three common cryptographic hash functions: MD5, SHA, and FSB Sasaki and Aoki (2009); Srinathan et al. (2007). Some of these functions are already broken and are no longer in use.

A MAC function, called a keyed hash function, can protect both a message's data integrity as well as its authenticity, so we requires that it must resist existential forgery under chosen-plaintext attacks (EF-CPA). This means that even if an adversary obtains some MACs for messages chosen by himself, it is unable to guess the MAC for a new message. The MAC is usually constructed from hash function (e.g. HMAC) or from block cipher algorithms (e.g. CBC-MAC) Bellare et al. (2000, 1996), as described in Table 2.

Another important tool is pseudorandom generator (PRG), which is widely applied anywhere in cryptography and some applications (i.e., gambling). Some simple methods, such as, linear feedback shift register (LFSR) and Linear congruential generators (LCG), have long been used as PRG for cryptosystem. Unfortunately, these methods are not secure for cryptographic applications. For example, an LFSR is a linear system, leading to fairly easy cryptanalysis; and LCG is not suitable for a Monte Carlo simulation because of the serial correlation. There exists some cryptographically secure PRG schemes, e.g., Blum Blum Shub (BBS) which has probable security under assumption of Quadratic residuosity problem Blum et al. (1986). However, this scheme is not appropriate for use in applications because it is very slow.

3. Background and Preliminaries

3.1. System and Construction Criteria

In this paper, we assume that a network device does not have a dedicated cryptographic hardware. In particular, it does not have a true random number generator. We assume that the device only has a short length (32 bits) accumulator and a 1-bit multiplication unit (it can be replaced by 1-bit accumulator), as well as a big enough memory. In addition, in MAC and PRG, we assume that there exists a secure storage unit which can prevent the disclosure of stored information. It is used to store the user's secret key and the current value of counter. Variable-length random function means that the output length of function can be changed in terms of user's requirements:

Definition 1 (Variable length random function) *A random function is a family of functions VL-RF: $\mathcal{D} \rightarrow \mathcal{R}$, where $\mathcal{D}$ is the message space with variable length, $\mathcal{R}$ is the range with variable desired length, and $|\mathcal{D}| \geq |\mathcal{R}|$.*

The output length $m = |\mathcal{R}|$ is an important parameter which related to the security of function. According to the birthday attack, the output length is at least $m = 2\kappa$-bit to remain collision resilient for a given secure parameter κ. *In this case the security of random function is enhanced against collision attack with the increase of κ.* For example, the random function with 341-bit output is more secure than that with 160-bit. Note that, *we choose an appropriate κ in terms of the security requirements of application.*

In addition, the value $\lambda = \max\{\frac{|\mathcal{R}|}{|\mathcal{D}|}\}$ is called compression ratio. Typically λ is less than 1. When $\lambda = 1$, a random function is called random permutation, which can be used to construct the encryption scheme. *Given a random function $f_k(x)$ with a certain λ, it is well-known that the output with $m \leq \lambda \cdot |R|$-bit holds good randomness and privacy for the input secret k in terms of HNP.* For instance, when $\lambda = 1/3$ and $|R| = 1024$-bit, the output with $m \leq 1024/3 = 341$-bit is random and secure against HNP attack. Further, λ is not a strict limit, e.g., a 381-bit output is also secure in this example if the ration is considered as $\lambda + \epsilon$, where ϵ is enough small.

We define the following quality criteria for a family of Hash/MAC/PRG functions intended for mobile devices and how they can be constructed:

Ease-of-realize: A family of Hash/MAC/PRG functions should be designed on a unified framework for simplifying the program coding. Our design principles are clear, so that anyone (for non-expert) can easily understand, realize, and use them without cryptography expertise.

Performance: The function value has to be calculated on common mobile devices which have a simple ALU and a modest memory. The performance is very critical because an execution point like a anti-spam (unsolicited commercial e-mail) filter has a limited amount of energy. Typically, only a small contingent of the resources are need to allocate for non-cryptographic algorithms and they has better performance than cryptographic algorithms.

Provable Security: It is required that the security of these functions follows from rigorous mathematical proofs, complexity theory and formal

reduction. Generally, those functions whose designs are based on a mathematical problem are called provably secure cryptographic functions. Moreover, these functions should have the basic security properties, e.g., the Hash/MAC function satisfies pre-image resistance, second pre-image resistance and collision resistance; and the PRG function is indistinguishable from truly random sequences.

Non-linearity and Unbiasedness: A good Hash/MAC/ PRG function should map the expected inputs as evenly as possible over its output range. This means that our construction must meet some basic cryptographic requirements, in which the non-linearity and unbiasedness properties are two well-known cryptographic criterions. A non-linear function f has a linear independency between input values x and y to output values, e.g. $f(x+y) \neq f(x)+f(y)$. Similarly, an unbiased function's output yields as many 0s as 1s over its input set. We follow these criterions in our construction.

3.2. Basic Mathematic Problems

We refer to $\{0,1\}^*$ as strings and $\{0,1\}^l$ as l-length strings. If x is a string or a number then $|x|$ denotes its length in bits. For $l \in \mathbb{N}$, we denote by 1^l the string of l "1" bits. Let $||$ denote concatenation. For sets X and Y, if $f : X \to Y$ is a function, then we call X the domain, Y the range, and the set $\{f(x)|x \in X\}$ the image of the function. An adversary is an algorithm. By convention, all algorithms are required to be efficient, meaning run in (expected) polynomial-time in the length of their inputs, and their running-time includes that of any overlying experiment.

- **Integer Factoring Problem.** Integer factorization is the decomposition of a composite number into smaller non-trivial divisors. That is, given $N = pq$, for any polynomial time (in $|N|$), and algorithm $\mathcal{A}$ and any polynomial $P(\cdot)$, for sufficiently large $|N|$, the factoring assumption holds that $\Pr[(p,q) = \mathcal{A}] \leq 1/P(|N|)$, where $1/P(|N|)$ denotes a negligible probability and $|N| = 1024$ is often chosen in practice.

- **Quadratic Residue Problem.** Let $N = pq$ be the public key and $p \equiv q \equiv 3 \pmod 4$. The Rabin function computes $y = x^2 \pmod N$ for $x \in \mathbb{Z}_N^*$. The square root problem says that given $y \in \mathbb{Z}_N$, without (p,q), to find x such that $y = x^2 \pmod N$ is as hard as factoring N. Note that there are four distinct roots x, including two group trivial roots ($\pm x$).

- **Hidden Number Problem.** The goal of square hidden number problem (SqHNP) Boneh et al. (2001) is to find a hidden number s, when given N and access to an oracle that on query $(x_1, \cdots, x_m)$ returns a value $(MSB_k((x+s)^2 \pmod N), \cdots, MSB_k((x_n+s)^2 \pmod N))$, where the operation $MSB_k(y)$ denotes the k most significant bits of y. The SqHNP assumption states that there is no polynomial time algorithm for this problem whenever $k = |N|/3$.

Given a prime $p \equiv 3 \pmod 4$, the square roots y modulo p can be computed by $x_p = y^{(p+1)/4} \pmod p$. Note that there is no square root for some numbers. When x_p and x_q are known, the square root problem can be resolved by Chinese remainder theorem. It has been proven that decoding the Rabin cryptosystem is equivalent to the integer factorization problem. This cryptosystem is provably secure (in a strong sense) against chosen plaintext attacks.

4. Construction of Hash Function

4.1. Square Hash and Its Shortcomings

A cryptographic hash function $Hash : \{0,1\}^* \to \{0,1\}^k$ is a function that takes an arbitrary block of data and returns a fixed-size bit string. A square hash function Etzel et al. (1999) is a cryptographic hash function with $\mathbb{Z}_N \to \mathbb{Z}_N$ for an enough large N, where it is hard to factorize N.

Definition 2 (Basic Construction) *The SQH family of hash functions from $\mathbb{Z}_N$ to $\mathbb{Z}_N$ is defined as: $\{f : \mathbb{Z}_N \to \mathbb{Z}_N\}$ where the functions f are defined as*

$$f(m) = m^2 \pmod N,$$

where, $N - N^{1/2} > m > N^{1/2}$ and $|N| = n$.

If $m < 2^{n/2}$, we have $m^2 < 2^n < N$, which means that $m^2 \pmod N = m^2$. In this case, given a hash value $y \in \mathbb{Z}_N$, we can use the split half method (or logarithmic search) to find m such that $m^2 = y$ because m^2 is a monotone increasing function. Similarly, another trivial root of $m^2 \pmod N$ is $N - m < N^{1/2}$ due to $(N-m)^2 = m^2 < N$.

This basic construction has the following shortcomings:

1. Assume that the length of N is n, the computation overhead of SQH function are $O(n^2)$. When n is large, the factorization of N is hard. But $O(n^2)$ is too large for frequent operations in mobile derives.

2. The output of SQH function is $|N|$ bits. Since we use SQH function with at less 1024-bits modulus size for secure integer factorization problem, the hash output size is at less 1024 bits. But the output of SHA-512/384 is just 512 or 384 bits with 1024-bit block size. Hence, the output of SQH function is too large for applications. Also, this feature is contradictory to the compression property of hash functions.

3. When m is too small (such as $m < N^{1/2}$), the output of SQH function is predictable in terms of above discussion. This predictability is considered as one such possible vulnerability that may be insecure for some applications.

We will focus on addressing these shortcomings by constructing more effective schemes in the remaining of this section.

4.2. Hash Function for $\mathbb{Z}_N \to \{0,1\}^k$

We present a new construction of square hash function to resolve the shortcomings in the basic construction as follows:

Definition 3 (Square Hash Construction) *Define a family of SqHash functions from* $\mathbb{Z}_N$ *to* $\{0,1\}^k$ *as:* $SqHash = \{f_{IV} : \mathbb{Z}_N \to \{0,1\}^k | IV \in \{0,1\}^l\}$ *where the function* f *is defined as*

$$f_{IV}(m) = MSB_k((m||IV)^2 \pmod N),$$

where, IV *denotes an initialization vector and* $MSB_k(t)$ *denotes* k *most significant bits of an integer* t*.*

Compared with the basic construction, SqHash has several advantages. Firstly, the output of SqHash is k bits in that $MSB_k()$ can be considered as a truncation function. For instance, k is 384 bits like SHA384. Next, let $m||IV = m \cdot 2^l + IV$. The initial vector IV increases the randomness of the output of SqHash function as a result of $(m \cdot 2^l + IV)^2 = m^2 \cdot 2^{2l} + IV^2 + m \cdot IV \cdot 2^{l+1} \pmod N$, especially for the item $m \cdot IV \cdot 2^{l+1}$. Moreover, this construction also helps avoid collision, that is, given a value y and a square

root m ($m^2 = y \pmod N$), we can check IV to determine whether this value is a valid pre-image of the four square roots of y.

Unfortunately, this construction does not reduce the computational complexity because we still need to perform the squaring operation. Hence, we make use of "modulo-$|N|$ circular convolution with carry bit" $m *_c m$ to replace the multiplication $m \cdot m$. The notation $*_c$ for cyclic convolution denotes convolution over the cyclic group of integers modulo $|N|$ Shamir (2008), that is,

$$(f *_c g)[k] = \sum_{i=0}^{n-1} f[i] \cdot g[(k-i) \mod n],$$

where, $n = |N|$ and $a[k]$ denotes the k-th bit of integer a.

In order to simulate the multiplication operation, we define circular convolution with carry bit as follows:

$$\begin{aligned} r[k] &= (f *_c g)[k] + c_{k-1} \pmod 2 \\ c_k &= ((f *_c g)[k] + c_{k-1} - r[k])/2 \end{aligned}$$

where, $r[k]$ is the k-th output bit, c_k denotes the k-th carry bit, and $c_0 = 0$. In Fig. 1, we show the difference between multiplication operation and circular convolution with carry bit. It is obvious that the latter is a more efficient process. Let $*'_c$ denote above circular convolution with carry bit. The SqHash function can be redefined as

$$f_{IV}(m) = MSB_k((m||IV) *'_c (m||IV)).$$

Note that we do not need module operations in this construction. Furthermore, another advantage of this construction is that we only need to calculate k most important bits without having to calculate other bits. For example, when $N = 1024$ bits and $k = 384$ bits, the computational overheads are 3/8 of those of multiplication operations. Note that the circular convolution does not change the nature of squaring operation in SqHash scheme. Thus, this transformation from squaring to convolution does not affect the security of SqHash scheme.

4.3. Hash Function for $\{0,1\}^* \rightarrow \{0,1\}^k$

A hash function must be able to process an arbitrary-length message into a fixed-length output. Usually, this can be achieved by breaking the input up into a series of equal-sized blocks, and operating on them in sequence using a

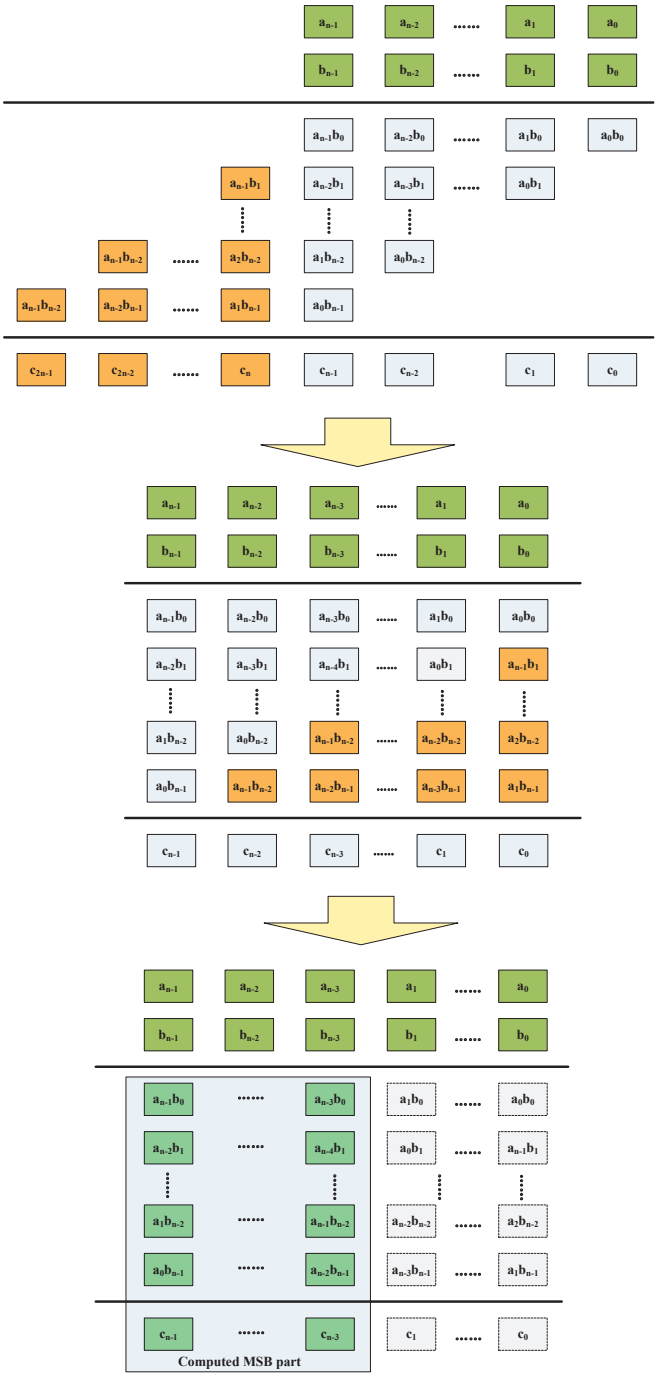

Figure 1: Multiplication and circular convolution.

one-way compression function. Hence, we define the hash function $\{0,1\}^* \to \{0,1\}^k$ based on the SQH function as follows: given $m = (m_n, \cdots, m_1, m_0)$,

$$\begin{aligned}
\sigma_0 &= MSB_k((m_0||IV) *'_c (m_0||IV)); \\
\cdots &\quad \cdots \\
\sigma_i &= MSB_k((m_i||\sigma_{i-1}) *'_c (m_i||\sigma_{i-1})); \\
\cdots &\quad \cdots \\
\sigma_n &= MSB_k((m_n||\sigma_{n-1}) *'_c (m_n||\sigma_{n-1})).
\end{aligned}$$

The final output of hash function is σ_n. The last processed block should also be unambiguously length padded. This is crucial to the security of our construction. Fig. 2 depicts such a construction. This kind of construction is also called Merkle-Damgärd construction Coron et al. (2005), in which any collision for the full hash function can be traced back to a collision in the compression function.

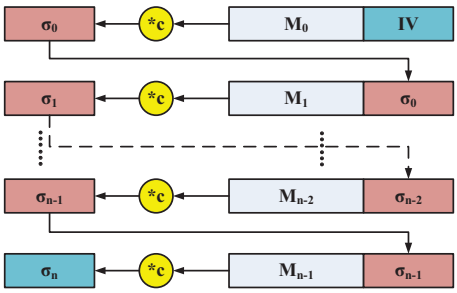

Figure 2: Hash construction for arbitrary-length messages.

5. Construction of MAC Function

Message authentication code (MAC), sometimes called keyed (cryptographic) hash function, specifies that an authenticated tag between two parties that share a secret key in order to validate message transmitted between these parties. The MAC value protects a message's data integrity and its authenticity by allowing verifiers (who also possess the secret key) to detect any changes in the message content. Typically, MACs are built from hash functions. Based on square function, we define the square MAC (in short SqMAC) function as follows:

Definition 4 (Square MAC Construction) *Let N be an integer to make the factorization attack difficult and $k \in \mathbb{N}$. A square MAC function from $\mathcal{M} \times \mathcal{K}$ to $\{0,1\}^k$ is defined as: $\{SqMAC_k : \mathbb{Z}_N \times \mathbb{Z}_N \to \{0,1\}^k\}$ where*

$$SqMAC_k(K, m) = MSB_k((m + K)^2 \pmod N),$$

and K is a secret key with $|K| = |N|$.

In this construction, $MSB_k()$ is necessary for ensuring security because, without this operation, the secret key K can be computed by

$$K = \frac{\sigma_2 - \sigma_1}{2(m_2 - m_1)} - \frac{m_1 + m_2}{2} \pmod N$$

if we have two pairs (m_1, σ_1) and (m_2, σ_2), where $\sigma_1 = (m_1 + k)^2$ and $\sigma_2 = (m_2 + x)^2 \pmod N$. In addition, we prove that the secret key K cannot be revealed in terms of Theorem 1 even if the adversary has observed a lot of message-MAC pairs.

We also make use of circular convolution to replace the square operation for obtaining higher efficiency, that is,

$$SqMAC_k(K, m) = MSB_k((m + K) *'_c (m + K)).$$

Next, we focus on the MAC construction for arbitrary-length message to be authenticated. HMAC is such a MAC. HMAC(K,m) is mathematically defined by

$$HMAC(K, m) = H((K \oplus opad)||H((K \oplus ipad)||m)),$$

where $H(\cdot)$ denotes a hash function. Based on this construction, we present a new MAC function by extending the function $SqMAC_k(\cdot)$ and above-mentioned hash function for $\{0,1\}^* \to \{0,1\}^k$, as follows:

1. To compute the hash value σ_n based on the algorithm in Section 4.3 but the initialization vector IV will be replaced by K; and

2. To output the MAC code by using $SqMAC_k(K, m')$ and $m' = K||\sigma_n$.

In this construction, $m' = K||\sigma_n$ is used to expand the length of authenticated message.

6. Construction of Pseudorandom Generator

A pseudorandom generator, abbreviated as PRG, is an efficiently-computable function which generates a sequence of $\{0, 1\}$ that approximates the properties of random string. Strictly speaking, no efficient algorithm can distinguish this sequence from a random sequence with significant advantage. Although there exists some practical PRG algorithms, such as, linear congruential generators (LCG), lagged Fibonacci generators (LFG), and linear feedback shift registers (LFSR), we present a fast PRG construction based on square problem with a simple counter.

6.1. Secure Square-PRG

In this subsection, we present a new PRG function on the modular square root (MSR) problem. Although we have constructed secure Hash and MAC scheme based on the MSR problem, it is still a challenging task for constructing secure PRG function due to its higher security requirements. Strictly speaking, PRG function can be constructed on the Hard-core predicate of a one-way function, but Hash/MAC function only needs the one-way property Berbain et al. (2006). Hence, we first focus on the Hard-core predicate of MSR problem.

To generate pseudorandom number, we assume that a secret x is stored in the device and a counter m_i is used to produce different output of PRG function for each PRG query, that is, $m_{i+1} = m_i + 1$ should be updated automatically. Note that the value m_i should be kept secret. Based on these assumptions, we define a one-way function as

$$f_x(m_i) = (x + m_i)^2 \pmod{N}.$$

Next, we find out the Hard-core predicate of this function according to the following theorem:

Theorem 1 *The two least significant bits (LSB) of next function output* $MSB_k(f_x(m_i))$ *can only be guessed with* $1/3$ *probability if* m_i *is an unknown counter.*

Proof. Let $m_{i+1} = m_i + 1$. Given two continuous values $f_x(m_i)$ and

$f_x(m_{i+1})$, we have the following equation

$$\begin{aligned} f_x(m_{i+2}) &= (x + m_i + 2)^2 \\ &= (x + m_i)^2 + 4(x + m_i) + 4 \\ &= 2(x + m_i + 1)^2 - (x + m_i)^2 + 2 \\ &= 2f_x(m_{i+1}) - f_x(m_i) + 2. \end{aligned}$$

Assume that e and e' are two $l = |N| - k$ bits numbers. Let $f_x(m_{i+1}) = MSB_k(f_x(m_{i+1})) \cdot 2^l + e \pmod N$ and $f_x(m_i) = MSB_k(f_x(m_i)) \cdot 2^l + e' \pmod N$, so that we have the following equation

$$\begin{aligned} MSB_k(f_x(m_{i+2})) &= MSB_k(2f_x(m_{i+1}) - f_x(m_i) + 2) \\ &= 2 \cdot MSB_k(f_x(m_{i+1})) - \\ &\quad MSB_k(f_x(m_i)) + \\ &\quad MSB_k(2e - e' + 2) \pmod N. \end{aligned}$$

In terms of $0 \le e, e' < 2^l$, we have $-2^l < 2e - e' + 2 < 2^{l+1}$. This means that

$$MSB_k(2e - e' + 2) = \begin{cases} -1 & 2e - e' + 2 < 0 \\ 0 & 0 \le 2e - e' + 2 < 2^l \\ +1 & 2^l \le 2e - e' + 2 \end{cases}$$

Hence, $MSB_k(f_x(m_{i+2}))$ has three possible values and each value can be guessed with the same probability $1/3$ according to the difference of $LSB_2(MSB_k(f_x(m_{i+2})))$.

This theorem indicates the Hard-core predicate of $f_x(m)$ is $LSB_2(MSB_k(f_x(m)))$. Further, it means that we can guess $2l$-bits in the l-th next value with the probability $\frac{1}{3^l}$. Hence, in order to improve efficiency, the successive $2l$ bits can be outputted $LSB_{2l}(MSB_k(f_x(m_i + l)))$, where m_i is the counter of previous output value. Thus, we define a new square PRG function as follows:

Definition 5 (Secure Square-PRG) *Given a counter i, a family of square PRG functions with $2l$-bits output is defined as: $\{SqPRG_{x,i,k} : 1^{2l} \to \{0,1\}^{2l} | x, i \in \mathbb{Z}_N, k \in \mathbb{N}\}$, where the function $SqPRG$ is defined as*

$$SqPRG_{x,i,k}(1^{2l}) = LSB_{2l}(MSB_k((x + i * l)^2 \pmod N)),$$

where, $MSB_k(t)$ and $LSB_k(t)$ denote k most and least significant bits of an integer t, the counter is updated to $i + l$ after this process, and $N - N^{1/2} > x > N^{1/2}$.

6.2. Efficient Implementation

To implement $SqPRG$ function, we can still use the "circular convolution with carry bit" to reduce the computational overheads, but we must deal with how to compute a particular bit (specially for LSB bits) correctly in $m^2 \pmod N$. More precisely, in order to be certain about the effect of the carry entering this bit (or LSB_l) position, we have to compute all earlier bits in the worst case. Fortunately, it is easy to show that the carry into each bit position in the computation of m^2 can be at most 11 bits long [1] for $|N|$ between 1024 and 2048. Thus, if we add $u = 32$ additional low order bits to the computed window, we have only a small (negligible) probability of less than $2^{11}/2^{32} = 1/2^{21}$ of computing an incorrect carry into the 33-th bit we compute. We call it "least significant bits with window" ($\widetilde{LSB}_l^u$). We present this process in Fig. 3.

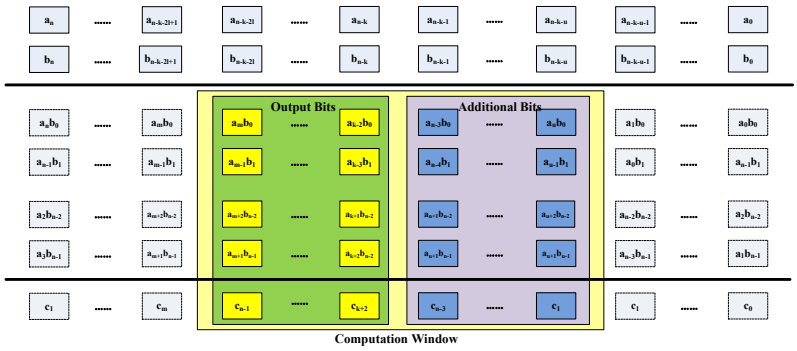

Figure 3: Circular convolution for least significant bits with window.

So that we can replace the above $SqPRG$ function by using

$$SqPRG_{x,i,k}(1^{2l}) = \widetilde{LSB}_{2l}^{u}((x + i * l) *_c (x + i * l)).$$

This can guarantee an extremely small error probability while keeping the average running time only slightly higher than always computing $l + 32$ bits.

[1] The value of carry bit is at most $2^{11} - 1$, so it can effect at most $\log_2(2^{11} - 1) \approx 11$ bits long.

7. Security Analysis

7.1. Security analysis of SqHash functions

The SQH function $f_{IV}(m)$ is a cryptographic hash function with preimage resistance and collision resistance Ouafi and Vaudenay (2009). The latter property also means that this function is second-preimage resistance. The preimage resistance stems from the fact that Square function $y = x^2 \pmod N$ is a trap-door one-way function in that given y, without (p, q), to find x such that $y = x^2 \pmod N$ is as hard as factoring N.

The collision resistance is based on the infeasibility of integer factoring problem Dubois et al. (2007). Assume that given y, there exist two values x_1 and x_2 for $(x_1||IV)^2 = (x_2||IV)^2 \pmod N$ and $x_1 \neq x_2$, such that we have $(x_1||IV)^2 - (x_2||IV)^2 = (x_1 - x_2)((x_1 + x_2)2^l + 2 \cdot IV) \cdot 2^l = 0 \pmod N$. This means that $N|(x_1 - x_2)((x_1 + x_2)2^{l-1} + IV)$. Thus, we can find a factor of N by using $\gcd((x_1 - x_2), N)$ only if not $N|((x_1 + x_2)2^{l-1} + IV)$. Otherwise, we can repeat this process to find out the factor of N. Further, given a fragment $MSB_k(y)$ of y, it is harder to find a valid x which includes IV, that is, $y = (x||IV)^2 \pmod N$. Even if with (p, q), it is hard to find a valid e with $y = MSB_k(y)||e$ such that $y = (x||IV)^2 \pmod N$.

7.2. Security analysis of SqMAC functions

Given the function $SqMAC_k(K, x)$, we prove that the secret key K cannot be revealed even if the adversary obtains sufficient message-MAC pairs. In this proof, we consider a hidden number problem: Let K be a random hidden element of $\mathbb{Z}_N$. We are given $N, k = |N|/3, |N| = n$, and $(x_i, MSB_k((x_i+K)^2 \pmod N))$ for random values $x_1, \cdots, x_t$. The problem is to find K. That is, we assume that $MSB_k((x_i + K)^2 \pmod N)) = y_i$, so that we have

$$x_i^2 + K^2 + 2x_iK = y_i \cdot 2^k + e_i \pmod N, \qquad i = 1, \cdots, t$$

where e_i are variables that correspond to unknown low order bits and $|e_i| \leq 2^{n-k} = 2^{2|N|/3}$. We are therefore forced to eliminate unknown K^2 from the above relation by using the equation:

$$x_i^2 - x_1^2 + 2(x_i - x_1)K = (y_i - y_1)2^k + (e_i - e_1) \pmod N.$$

Next, we also eliminate K by the equation:

$$\begin{aligned}
&(x_i^2 - x_1^2)(x_i - x_1) - (x_j^2 - x_1^2)(x_j - x_1) \\
&= (x_i - x_1)(x_j - x_1)(x_i - x_j) \\
&= (x_i - x_1)(y_i - y_1)2^k - (x_j - x_1)(y_j - y_1)2^k + \\
&\quad (x_i - x_1)(e_i - e_1) - (x_j - x_1)(e_j - e_1) \pmod N.
\end{aligned}$$

Also, we rewrite this equation as a polynomial in the unknown e_i, e_j, e_1, namely:

$$f_i(e_i, e_j, e_1) := A_{i,1}e_i + B_{j,1}e_j + C_{i,j}e_1 - X_{i,j} \pmod N.$$

where $A_i = x_i - x_1$, $B_j = x_1 - x_j$, $C_j = x_j - x_i$, and $X_i = ((x_i - x_1)(y_i - y_1) - (x_j - x_1)(y_j - y_1))2^k - (x_i - x_1)(x_j - x_1)(x_i - x_j)$. Based on this function, we setup a lattice of dimension $2t - 1$ as a real matrix M:

$$\begin{pmatrix}
1 & 0 & 0 & 0 & \cdots & 0 & X_{3,2} & \cdots & X_{t,2} \\
0 & 2^{k-n} & 0 & 0 & \cdots & 0 & C_{3,2} & \cdots & C_{t,2} \\
0 & 0 & 2^{k-n} & 0 & \cdots & 0 & B_{2,1} & \cdots & B_{2,1} \\
0 & 0 & 0 & 2^{k-n} & \cdots & 0 & A_{3,1} & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \cdots & 2^{k-n} & 0 & \cdots & A_{t,1} \\
0 & 0 & 0 & 0 & \cdots & 0 & N & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & N
\end{pmatrix}$$

Next, we set $v = (1, e_1, e_2, \cdots, e_t, k_2, \cdots, k_t)$. It follows that for this integer vector v we get:

$$v \cdot M = (1, \frac{e_1}{2^{n-k}}, \cdots, \frac{e_t}{2^{n-k}}, 0, \cdots, 0).$$

Thus, the lattice point $v \cdot M$ has only $t + 1$ non-zero entries, and each of these is less than 1. And its Euclidean norm is less than $\sqrt{t+1}$. On the other hand, it is easy to see that the determinant of the lattice $L(M)$ equals $N^{t-2} \cdot 2^{(k-n)t}$. In addition, making using of the Gaussian heuristic for short lattices vectors, we expect that our vector is the shortest point in this lattice $L(M)$ as long as $\sqrt{t+1} \ll \sqrt{2t-1}(N^{t-2} \cdot 2^{(k-n)t})^{1/(2t-1)}$. Let $N \approx 2^n$ and ignoring low-order terms, this condition is simplified to $\sqrt{\frac{t+1}{2t-1}} \approx 2^{-1/2} \ll 2^{k/2-n/t} < 2^{k/2} = 2^{n/6}$. Therefore, the Lattice-based reduction methods cannot find the secret K Boneh and Venkatesan (1997).

7.3. Security analysis for SqPRG function

The security requirement for PRG function is that the output of pseudorandom sequences should be computationally indistinguishable from truly random sequences. For practicality, such pseudorandom generators can be constructed easily on the hard-core predicates. In fact, we have proved that our SqPRG function is constructed on the hard-core predicates with the probability $1/3^l$ for $2l$-bits outputs in Theorem 1.

8. Improved Algorithms

We use two different approaches (i.e., the truncated function and circular convolution) to improve the performance of all proposed schemes. Usually, long-size cryptographic algorithms are not suitable for providing security on wireless devices due to their limited computation and communication capabilities. In a cryptosystem, key length is usually much longer than the "word size" of mobile devices, typically 16 or 32 bits. Using short ALU to deal with long data often leads to more complex programs for cryptography algorithms. Moreover, traditional Hash/MAC/PRG functions do not contain algebra operations thus ALU computing power would normally not be fully utilized. However, the algorithms proposed in this paper are easy to implement, and can take advantage of ALU computing power. We will give basic construction on MSB and convolution, on which SqHash/SqMAC/SqPRG can be easily constructed.

First, our basic construction (called "Convol-MSB" algorithm in Fig.4) can be implemented easily on mobile devices. We improve the computation by replacing integer multiplication with circular convolution. The advantage of this method is that each output bit is calculated from right to left, while traditional multiplication is from top to bottom, then right to left (see Fig. 1). In addition, we also use MSB operation (with window) to further reduce computation overheads. Based on them, the computation complexity of "Convol-MSB" is $O(kn)$ instead of $O(n^2)$ in the traditional multiplication way. This is equal to about k/n of the original overhead.

The basic "Convol-MSB" algorithm is constructed on bit-AND operator. Thus, it does not take full advantage of the ALU computing resources. Taking into account of bit-parallel processors in ALU hardware, we improve the above-mentioned algorithm to a new "Convol-MSB-IMP" algorithm in Fig. 5. In this algorithm, we make use of the algorithm $CyclicLeftShift(x, k)$ to cycles k positions to the left for the elements in x. Furthermore, we use $a * .b$

Table 3: Comparisons among SqHash, SqMAC, SqPRG and their improved schemes.

Name	Item	Description
SqHash	Equation	$MSB_k((m\|\|IV)^2 \pmod N)$
	Performance	$O(n^2)$
	Improved Equation	$MSB_k((m\|\|IV) *'_c (m\|\|IV))$
	Performance	$O(n*k)$
SqMAC	Equation	$MSB_k((m+K)^2 \pmod N)$
	Performance	$O(n^2)$
	Improved Equation	$MSB_k((m+K) *'_c (m+K))$
	Performance	$O(n*k)$
SqPRG	Equation	$LSB_{2l}(MSB_k((x+i+l)^2 \pmod N))$
	Performance	$O(n^2)$
	Improved Equation	$\widetilde{LSB}^u_{2l}((x+i+l) *'_c (x+i+l))$
	Performance	$O(n*(2l+u))$

to give the bitwise AND of a and b, and $Sum(x)$ to give the total number of elements in list x. Based on these improvements, the computation overhead is further reduced to $O(kn/w)$ since the actual scale is the word size of mobile devices w in each parallel processing. Note that above pseudo-codes are not real ALU calls and we need to expand these calls according to actual input length.

In Table 3, we give a summary of the performance estimate of SqHash, SqMAC, and SqPRG compared with corresponding improved schemes. Obviously, the improved scheme achieves a computational overhead ($O(nk)$) smaller than the original schemes ($O(n^2)$). The ratio between them is about $k/n = O(nk)/O(n^2)$. This result is benefited from the use of truncated function MSB or LSB, as well as circular convolution operation. In addition, these truncated functions also increase the difficulty of attacks against the improved schemes. Moreover, the overhead $O(nk)$ means that the scheme is less sensitive than the original scheme for data length n. For example, when we want to double the length of processed message, the overhead of improved schemes will increase by about 100% (from $O(nk)$ to $O(2nk)$), but in the original schemes those will increase by about 400% (from $O(n^2)$ to $O(4n^2)$).

To validate the efficiency of our approach, we implemented several algorithms in the Mathematica 7.0 environment. In Fig. 6, we show a compar-

Algorithm Convol-MSB(x, n, k)
Require: x is an input integer, n is the length of x, and k is the length of output result.
$s = 0$;
for $i = n - k + 1$ to n **do**
  for $j = 1$ to n **do**
    $r = x[j] \cdot x[(i - j) \pmod n]$;
    $s = s + r$;
  end for
  $idx = i - n + k$;
  $r[idx] = s \pmod 2$;
  $s = (s - r[idx])/2$;
end for
Return r;

Figure 4: Basic convolution-MSB algorithm.

Algorithm Convol-MSB-IMP(x, n, k)
Require: x is an input integer, n is the length of x, and k is the length of output result.
$t_x = \text{CyclicLeftShift}(x, n - k)$;
$s = 0$;
for $i = 1$ to k **do**
  $t_x = \text{CyclicLeftShift}(x, 1)$;
  $t = x * .t_x$;
  $r[i] = sum(t + s) \pmod 2$;
  $s = (sum(t + s) - r[i])/2$;
end for
Return r;

Figure 5: Improved convolution-MSB algorithm.

ison of experimental results among Square, convolution-MSB and improved convolution-MSB algorithms. In this figure, the total length of N is changed from 512 to 1024-bits and the size of output results is about 1/3 of length of N. The computation cost is proportional to $|N|$ for the square algorithm and the convolution-MSB algorithm and the length of N has a greater impact on the Square algorithm. However, the computation cost is independent to the length of N in the improved convolution-MSB algorithm. In summary, the

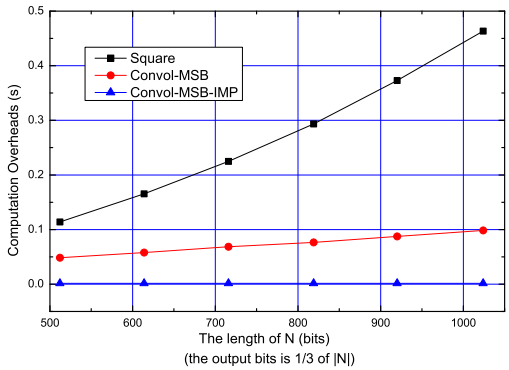

Figure 6: Comparison of experimental results among Square, convolution-MSB and improved convolution-MSB algorithms.

experimental results show that our improved convolution-MSB method has better performance than general square algorithm.

Table 4: The result of variable length experiments

No.	ratio	cycles	seconds	No.	ratio	cycles	seconds
1	10/32	0.003	0.000001	2	21/64	0.006	0.000002
3	32/96	0.010	0.000005	4	42/128	0.017	0.000007
5	53/160	0.022	0.000010	6	64/192	0.030	0.000015
7	74/224	0.040	0.000019	8	85/256	0.050	0.000024
9	96/288	0.065	0.000032	10	106/320	0.076	0.000037
11	117/352	0.092	0.000045	12	128/384	0.107	0.000053
13	138/416	0.123	0.000061	14	149/448	0.152	0.000075
15	160/480	0.158	0.000078	16	170/512	0.181	0.000090
17	181/544	0.201	0.000100	18	192/576	0.234	0.000117
19	202/608	0.258	0.000128	20	213/640	0.275	0.000137
21	224/672	0.310	0.000154	22	234/704	0.347	0.000174
23	245/736	0.374	0.000186	24	256/768	0.408	0.000204
25	266/800	0.436	0.000218	26	256/832	0.463	0.000231
27	288/864	0.496	0.000248	28	298/896	0.530	0.000265
29	309/928	0.587	0.000293	30	320/960	0.607	0.000295
31	330/992	0.678	0.000347	32	341/1024	0.694	0.000347

To evaluate the performance, we use Mathematica7 and C language to realize our schemes. The median runtime of SqHash (—N—=1024 and k=384) is 0.0065 seconds/operation for Mathematica 7 and 0.000381 second-

s/operation for VC 2005 in a 32-bit Window 7 system. The median number of CPU cycle counting is about 0.79 Millicycles/operation. We find that our SqHash, SqMAC, SqPRG schemes and some existing standards have the almost same overheads if the benefits of assembly language are not considered. In addition, the SqHash has a concise and clear programm coding: 40 lines for Mathematica and 150 lines for C language.

Further, we evaluate the performance of variable-length output for one operation. In this experiment, the input length of SqHash is changed from 32 bits to 1024 bits, and it generates an output in accordance with the ratio of 1/3, i.e., changed from 10 bits to 341 bits. The results, including median CPU cycle counting and median runtime, are shown in Table 4. It is easy to see that computational complexity appears to grow linearly with the output length. The SqHash not only has similar performance with non-cryptographic schemes for a shorter output, but also has good performance with cryptographic schemes for a longer output.

9. Statistical Tests

The security proofs have shown that our schemes are able to resist some common attacks, but this does not mean that they have excellent properties to meet the requirements of various applications. To end it, a set of statistical tests for randomness are described in this section. In our statistical tests, the performance of our schemes and SHA-2 functions is measured by several valuation tools written in Mathematica 7 and VC 2005. These tools are run on a IBM desktop PC (2.0 GHz and 2 GB RAM) with Windows 7 OS.

9.1. Entropy Testing

Shannon entropy is a common measure of the uncertainty associated with a random variable. In cryptanalysis, entropy is often roughly used as a measure of the unpredictability of random sequences, e.g., a cryptographic key. For a "true" random sequences, "0" and "1" both are statistically independent and have equal probability 1/2, hence the entropy rate of the sequence is 1 bit per character. In Figure 7(a), we show the measurement results of one-bit entropy testings in three schemes: SHA-2 (SHA-256), SQH and SqHash functions. The former has the 256-bit output size and the latter two are 384 bits. In this experiment one can observe that the entropy rate of sequence tends to 1 bit per character (0/1) when the length of three sequences is increased from 1,024 to 10,240 bits. It is easy to find that the

one-bit randomness of SqHash function is basically the same as those of SQH and SHA-2 functions. More accurately, Figure 8 (a) shows the measurement results of 8-bit entropy testings: the entropy tends to 7.9 with increase of sequence length, that is, the entropy rate tends to 0.98 bit per character (0/1). This means that the 8-bit randomness of SqHash function is basically the same as those of SQH and SHA-2 functions. To extend this result, a n-bit key that is randomly generated has approximate n bits of entropy. It takes (on average) 2^{n-1} guesses to break by brute force.

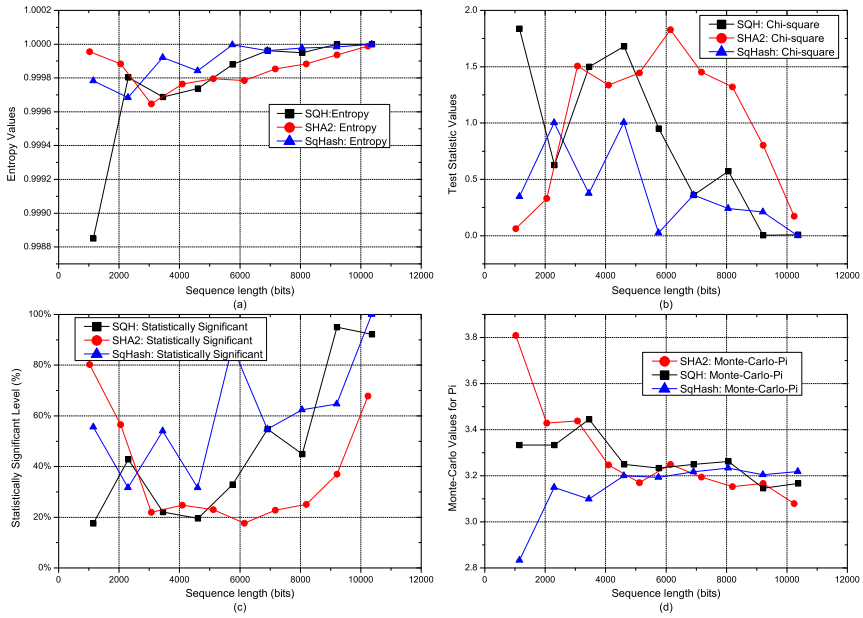

Figure 7: The comparison result of one-bit statistical tests for SHA-2 (256), SQH, SqHash functions.

9.2. Chi-Squared Testing

The unbiasedness of random functions can be tested by Chi-Squared independence test. The Chi-Squared test is the most commonly used test for the randomness of data, and is extremely sensitive to errors in pseudorandom sequence generators. In essence, the Chi-Squared test, belongs to a statistical hypothesis testing, is a conclusion-generation procedure that has two possible outcomes, either accept null hypothesis (H_0: the sequence is random) or accept the alternative hypothesis (H_1 the sequence is non-random). For each applied test, a decision or conclusion is derived that accepts or rejects the

null hypothesis, i.e., whether the detected function is (or is not) producing random values, based on the sequence that was produced.

For a m-bit block test on a n-block length sequence $\{X\}$, the test statistic S_m is defined as the sum of deviations between expected (e_i, asserted by the null hypothesis) and observed simultaneous frequencies (o_i) of a detected sequence:

$$S_m = \sum_{i=0}^{2^m-1} \frac{(o_i - e_i)^2}{e_i} = \sum_{i=0}^{2^m-1} \frac{(o_i - np_i)^2}{np_i},$$

where $e_i = np_i$ and $p_i = \Pr[X = i] = \frac{1}{2^m}$ for null hypothesis. This equation tests a null hypothesis stating that the frequency distribution of certain events observed in a sample is consistent with a theoretical uniform distribution.

In Figure 7 (b), we show the results of one-bit Chi-Squared tests (m=1) where the length of SHA-2, SQH, SqHash sequences is increased from 1,024 to 10,240 bits. In this case, the expected probability is $p_0 = p_1 = 1/2$ in null hypothesis. The test statistic of SHA-2 (SHA-256) is ranged from 0.06 to 1.82; that of SQH is from 0.01 to 1.83; and that of SqHash is from 0 to 1.01. Further, Figure 8 (b) shows the similar results in 8-bit Chi-Squared tests (m=8), where the expected probability is $p_i = 1/256$ in null hypothesis for $i = 0, \cdots, 255$. Since the test statistic S_m is in essence a square cumulative of deviations of e_i and o_i, this means that the smaller the value of S_m, the better the randomness of the detected function. Hence, our experiments show that the output of SqHash has good average-case randomness than those of SQH and SHA-2.

9.3. Significance Testing

For each Chi-Squared test, a statistical significance testing has been used to determine the acceptance or rejection of the randomness (null) hypothesis H_0. In this test, the amount of evidence, required to accept that an event is unlikely to have arisen by chance, is known as the significance level α. To achieve the hypothesis testing, the test statistic value S_m on data, called P-value, should be compared to the critical value α: if P-value does not exceed the critical value α ($P-value < \alpha$), the null hypothesis (H_0) for randomness is rejected. Otherwise ($P - value \geq \alpha$), the null hypothesis is not rejected (i.e., H_0 is accepted). Typically, α is chosen in 1%: A $P-value \geq 0.01$ would mean that the sequence would be considered to be random with a confidence of 99%.

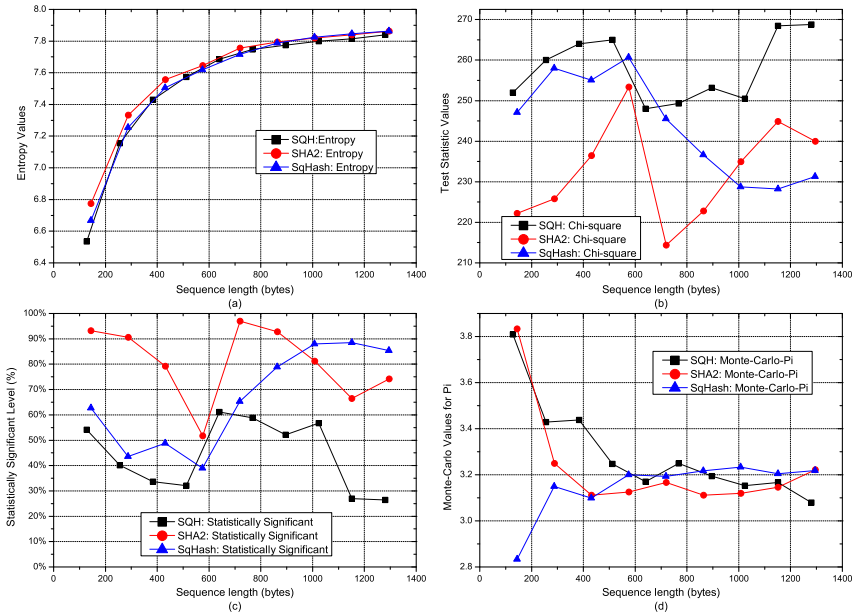

Figure 8: The comparison result of one-byte statistical tests for SHA-2 (256), SQH, SqHash functions.

In Figure 7 (c), we show the result of significance testings from the test statistic values S_m on Figure 7 (b). Here, the P-value is calculated by comparing the statistic value S_m to a Chi-Squared distribution (χ^2) [2]. The range of P-value of SHA-2 is from 17.63% to 80.26%; the range of SQH is from 17.53% to 95.02%; and the range of $SqHash$ is from 80.26% to 99.99%. The similar results is also shown in Figure 8 (c). According to the criterion $P - value \geq 0.01$, three functions would be considered to be random. More importantly, the randomness of SqHash has the better randomness than those of SQH and SHA-2 due to the P-values of SqHash is much larger than those of SQH and SHA-2.

9.4. Monte-Carlo Testing

Monte-Carlo simulation is one of the main applications involving the use of random number generators. It is also one of the best methods of testing

[2]Given a n-block sequence and the statistic value S_m, we define $P - value = Q(n/2, S_m/2) = \int_{S_m/2}^{\infty} t^{n/2-1}e^{-t}dt/\Gamma(n/2)$, where $\Gamma(x) = \int_0^{\infty} t^{x-1}e^{-t}dt$ and $Q(s,t)$ is the incomplete gamma function.

the randomness properties of such generators, by comparing results of simulations using different generators with each other, or with analytic results. One practical way to test a random number generator is to use it for Monte-Carlo simulation of two dimensional applied to approximating the value of π. 1) consider a circle inscribed in a unit square, where the circle and the square have a ratio of areas that is $\pi/4$; 2) each successive sequence of six bytes is used as 24 bit X and Y co-ordinates within the square. If the distance of the randomly-generated point is less than the radius of a circle inscribed within the square, the six-byte sequence is considered a "hit"; 3) count the number of points inside the circle and the total number of points; 4) the ratio of the two counts is an estimate of the ratio of the two areas, which is $\pi/4$, and multiply the result by 4 to estimate π.

The results of Monte-Carlo testing are shown in Figure 7 (d) and 8 (d). We use the nonlinear (Gauss) curve fit methods ($y = a_0 + a_1 x^b$) to estimate the curve parameters in 5. While the parameter a_0 is closer to π, the function is more suitable for simulation tests. Hence, the SqHash provides a good approximation for π in Monte-Carlo simulation.

Table 5: Curve Fit for Monte Carlo Testing

	a_0	a_1	b
SHA-2	2.947	0.8546	-0.6805
SQH	3.368	-0.007675	1.525
SqHash	3.224	-0.3865	-1.745

10. Conclusions

In this paper, we prompt the idea of constructing various basic cryptographic primitives (such as hash, MAC, and PRG) from a common core algorithm to simplify the management and improve the efficiency of current cryptographic implementation practices on mobile devices. For this purpose, we present the design of a family of cryptographic primitives based on the common core squaring operation. Our design takes advantage of the practical construction in ALU hardware, such as parallel processing units or algebra operation units, so that the proposed schemes can be efficiently realized on resource-constrained mobile devices. Moreover, the proposed schemes are provably secure under the hidden number problem and hard-core predicate.

References

Appleby, A., 2011. Murmurhash. hash. https://sites.google.com/site/murmurhash/.

Bellare, M., Canetti, R., Krawczyk, H., 1996. Keying hash functions for message authentication. In: Koblitz, N. (Ed.), CRYPTO. Vol. 1109 of Lecture Notes in Computer Science. Springer, pp. 1–15.

Bellare, M., Kilian, J., Rogaway, P., 2000. The security of the cipher block chaining message authentication code. J. Comput. Syst. Sci. 61 (3), 362–399.

Berbain, C., Gilbert, H., Patarin, J., 2006. Quad: A practical stream cipher with provable security. In: Vaudenay, S. (Ed.), EUROCRYPT. Vol. 4004 of Lecture Notes in Computer Science. Springer, pp. 109–128.

Blum, L., Blum, M., Shub, M., 1986. A simple unpredictable pseudo-random number generator. SIAM J. Comput. 15 (2), 364–383.

Boneh, D., Halevi, S., Howgrave-Graham, N., 2001. The modular inversion hidden number problem. In: Boyd, C. (Ed.), ASIACRYPT. Vol. 2248 of Lecture Notes in Computer Science. Springer, pp. 36–51.

Boneh, D., Venkatesan, R., 1997. Rounding in lattices and its cryptographic applications. In: Saks, M. E. (Ed.), SODA. ACM/SIAM, pp. 675–681.

Coron, J.-S., Dodis, Y., Malinaud, C., Puniya, P., 2005. Merkle-damgård revisited: How to construct a hash function. In: Shoup, V. (Ed.), CRYPTO. Vol. 3621 of Lecture Notes in Computer Science. Springer, pp. 430–448.

Dubois, V., Fouque, P.-A., Shamir, A., Stern, J., 2007. Practical cryptanalysis of sflash. In: Menezes, A. (Ed.), CRYPTO. Vol. 4622 of Lecture Notes in Computer Science. Springer, pp. 1–12.

Etzel, M., Patel, S., Ramzan, Z., 1999. Square hash: Fast message authenication via optimized universal hash functions. In: Wiener, M. J. (Ed.), CRYPTO. Vol. 1666 of Lecture Notes in Computer Science. Springer, pp. 234–251.

Fowler, G., Vo, P., Noll, L. C., 1988. Fnv hash. http://www.isthe.com/chongo/tech/comp/fnv/index.html.

Jenkins, B., 2009. Jenkins hash. http://www.burtleburtle.net/bob/hash/doobs.html.

Kiltz, E., 2001. A primitive for proving the security of every bit and about universal hash functions & hard core bits. In: Freivalds, R. (Ed.), FCT. Vol. 2138 of Lecture Notes in Computer Science. Springer, pp. 388–391.

Ouafi, K., Vaudenay, S., 2009. Smashing squash-0. In: EUROCRYPT. pp. 300–312.

Sasaki, Y., Aoki, K., 2009. Finding preimages in full md5 faster than exhaustive search. In: EUROCRYPT. pp. 134–152.

Shamir, A., 2008. Squash - a new mac with provable security properties for highly constrained devices such as rfid tags. In: Nyberg, K. (Ed.), FSE. Vol. 5086 of Lecture Notes in Computer Science. Springer, pp. 144–157.

Srinathan, K., Rangan, C. P., Yung, M. (Eds.), 2007. Progress in Cryptology - INDOCRYPT 2007. Vol. 4859 of Lecture Notes in Computer Science. Springer.

Wang, X., Yu, H., 2005. How to break MD5 and other hash functions. In: Eurocrypt 2005.