A Pseudorandom Number Generator with KECCAK Hash Function

A. Gholipour and S. Mirzakuchaki

Abstract—**This paper presents a pseudorandom generation algorithm, which is based on the KECCAK hash function and can pass the random test of the NIST (National Institute of Standards and Technology) Statistical Test Suite. Its security shows that can be utilized to generate pseudorandom bit sequences, which the experimental results show the KECCAK hash function has excellent pseudo randomness.**

Index Terms—**Pseudorandom number generator, KECCAK hash function, national Institute of Standards and technology statistical test suite**

I. Introduction

The security of many cryptographic mechanisms that are used in TCPA depends upon the generation of unpredictable quantities [1]. Example include the primes in the RSA encryption and digital signature schemes, the secret key in the DES and 3DES encryption algorithms, and the nonce used in challenge-response integrity-checking system. In all these cases, the quantities generated must be of sufficient size and be random in the sense that the probability of any particular value being selected must be sufficiently small to preclude an adversary from gaining advantage through optimizing a search strategy based on such probability.

Ideally, secrets required in cryptographic algorithms and protocols should be generated with a true random bit generator[2]. However, the generation of random bits is an inefficient procedure in most practical environments. Moreover, it may be impractical to securely store and transmit a large number of random bits if these are required in application such as the on-time pad. In such situations, substituting a random bit generator with a pseudo-random bit generator can ameliorate the problem.

A pseudo random bit generator is a deterministic algorithm which, given a binary sequence of length k, outputs a binary sequence of length l>>k which "appears" to be random. The input of the generator is called a pseudo random bit sequence.

Pseudo random bit generation can use a hash function f by first selecting a random seed s and then applying the function to the sequence of values s+1, s+2, … . The output sequence is f(s), f(s+1), f(s+2)… Depending on the properties of the one-way function used, it may be necessary to keep only a few bit of the output values f(s+i) in order to remove possible correlation between successive values [3].

Manuscript received June 21, 2011; revised November 4, 2011.

A. Gholipour is with the Iran University of Science and Technology, (e-mail: a_gholipour@elec.iust.ac.ir).

S. Mirzakuchaki is with the Department of Electrical Engineering, Iran University of Science and Technology, (e-mail: m_kuchaki@iust.ac.ir).

Probably the most popular hash functions in use today are MD5 and SHA-x algorithms. However, recent cryptanalytic advances have shown weaknesses that allow collisions to be computed for these hash functions much faster than by brute force [4-6]. Consequently, some pseudorandom number generators based on hash functions, which have been standardized by NIST (National Institute of Standards and Technology) in the U.S. are no longer secure.

Following the weakening of the widely-used SHA-1 hash algorithm and concerns over the similarly-structured algorithms of the SHA-2 family, the NIST has set up the SHA-3 competition with the goal of identifying one (or more) modern hash functions which can act as a drop in replacement for the SHA-2 family[7].

KECCAK hash function is one of these candidates accepted by NIST for the SHA-3 hash function competition. In this paper we present a new pseudorandom number generator based on the KECCAK hash function.

II. KECCAK Algorithm Description

KECCAK is a family of hash functions that are based on the sponge construction and use as a building block a permutation from a set of 7 permutations. There are 7 KECCAK-f permutations, indicated by KECCAK-f[b], where $b = 25 \times 2^l$ and l ranges from 0 to 6. KECCAK-f[b] is a permutation over $S \in Z_2^b$, where the bits of s are numbered from 0 to b - 1. b is the width of the permutation. These KECCAK-f permutations are iterated constructions consisting of a sequence of almost identical rounds. The number of rounds nr depends on the permutation width, and is given by $n_r = 12 + 2l$, where $2l = b/25$. This gives 24 rounds for KECCAK-f[1600].

The KECCAK Hash function produces a final digest message of 256 bits, which is dependent on the input message, composed of multiple blocks of 1024 bits each. The input message block is XORed onto a part of the current state and the result is passed through the KECCAK-f permutation. The KECCAK algorithm consists of 3 stages: (i) initialization and padding; (ii) absorbing phase; and (iii) squeezing phase. A pseudo code for this algorithm is depicted below[8,9].

KECCAK[r, c, d](M)

- Initialization and padding

  $S[x, y] = 0 \quad \forall (x, y) \quad in \quad (0...4,0...4)$

  $P = M \,||\, 0x01 \,||\, byte(d) \,||\, byte(r/8) \,||\, 0x01 \,||$

  $... \,||\, 0x00$

- Absorbing phase

  for every bolck P_i *in* P

$S[x,y] = S[x,y] \oplus P_i[x+5y],$

$\forall (x,y)\ \ such\ that\ \ x+5y \prec r/w$

$S = KECCAK\text{-}f[r+c](S)$

- Squeezing phase

While output is requested

$Z = Z \,||\, S[x,y],$

$\forall (x,y)\ \ such\ that\ \ x+5y \prec r/w$

$S = KECCAK - f[r+c](S)$

return Z

The state is logically grouped into a 5×5 matrix of 64-bit words. The KECCAK-f permutation consists of 24 rounds, which are identical except for the addition of a round-dependent constant. Each round has five steps (θ, ρ, π, χ and τ), which feature simple logical operations and permutations of the state bits. The initial state is all zero and in each round the introduced data is mixed with the current state.

$$R = \iota \ o\ \ \chi\ \ o\ \ \pi\ \ o\ \ \rho\ \ o\ \ \theta$$

$$\theta: \quad C[x] = A[x,0] \oplus A[x,1] \oplus A[x,2] \oplus A[x,3] \oplus A[x,4] \qquad \forall x\ in\ 0...4$$

$$D[x] = C[x-1] \oplus ROT(C[x+1],1)$$

$$\forall x\ in\ 0...4$$

$$A[x,y] = A[x,y] \oplus D[x]$$

$$\forall (x,y) in\ (0...4,0....4)$$

$$\rho: \quad A[x,y] = ROT(a[x,y], r(x,y))$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

$$\pi: \quad A[X,Y] = a[x,y]$$

$$\begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

$$\chi: \quad A[x,y] = B[x,y] \oplus (NOT\ \ B[x+1,y]\ \ AND\ \ B[x+2,y])$$

$$\forall (x,y) in\ (0...4,0....4)$$

$$\tau: \quad A[0,0] = A[0,0] \oplus RC$$

Here the following conventions are in use. All the operations on the indices are done modulo 5. A denotes the complete permutation state array and A[x, y] denotes a particular lane in that state. B[x, y], C[x] and D[x] are intermediate variables. The symbol ⊕ denotes the bitwise exclusive OR, NOT the bitwise complement and AND the bitwise AND operation. Finally, ROT(W, r) denotes the bitwise cyclic shift operation, moving bit at position i into position i + r (modulo the lane size).

The constants r(x,y) are the cyclic shift offsets and are specified in the following table.

The constants RC[i] are the round constants. The following table specifies their values in hexadecimal notation for lane size 64 and shown in table II.

III. Pseudorandom Bit Generation

The proposed pseudo-random number generator is based on the following elements [10]:

1. An initial seed s consisting of a k-bit string
2. A *basic* function *B* to obtain a k-bit string from another k-bit string
3. A (non-cryptographic) hash function *H*

We use the KECCAK as a one-way function *H* and use it to generate pseudorandom bit sequences by first selecting a random seed *s*, and then applying the function to the sequence of values *s*, *s* +1, *s* + 2,… , and the output sequence is *H*(*s*), *H*(*s* +1), *H*(*s* + 2),…. We use the KECCAK-f[1600] which the number of its input is 1024 bit and the output is 256.

The algorithm for generate pseudorandom bit sequence similar to the FIPS 186 one-way function using SHA-1 and shown below [11]:

TABLE II: VALUE OF RC[I] CONSTANT

RC[0] 0x0000000000000001	RC[12] 0x000000008000808B
RC[1] 0x0000000000008082	RC[13] 0x800000000000008B
RC[2] 0x800000000000808A	RC[14] 0x8000000000008089
RC[3] 0x8000000080008000	RC[15] 0x8000000000008002
RC[4] 0x000000000000808B	RC[16] 0x800000000000808B
RC[5] 0x0000000080000001	RC[17] 0x8000000000000080
RC[6] 0x8000000080008081	RC[18] 0x000000000000800A
RC[7] 0x8000000000008081	RC[19] 0x800000008000000A
RC[8] 0x000000000000008A	RC[20] 0x8000000080008081
RC[9] 0x0000000000000088	RC[21] 0x8000000000008080
RC[10] 0x0000000080008082	RC[22] 0x0000000080000001
RC[11] 0x000000080000000A	RC[23] 0x8000000800008008

Algorithm 1: Pseudorandom bit generator for KECCAK
INPUT: a 256-bit string Seed

OUTPUT: a 256L-bit string denoted $H_0(seed),...,H_L(seed)$

1. Pad Seed with KECCAK's padding method to obtain a 1024-bit message block t.
2. Break up t into 64 bit blocks: $t = m_1 \,||\, m_2 \,||\, ... \,||\, m_{16}$.
3. Execute KECCAK's absorbing phase.
4. Go to step 1.
5. The output is $H_0(seed),...,H_L(seed)$

TABLE I: VALUE OF OFFSET IN ρ STEP

	x=3	x=4	x=0	x=1	x=2
y=2	153	231	3	10	171
y=1	55	276	36	300	6
y=0	28	91	0	1	190
y=4	120	78	210	66	253
y=3	21	136	105	45	15

IV. Statistical Tests

Various statistical tests can be applied to a sequence to attempt to compare and evaluate the sequence to a truly random sequence. Randomness is a probabilistic property; that is, the properties of a random sequence can be characterized and described in terms of probability. The likely outcome of statistical tests, when applied to a truly

random sequence, is known a priori and can be described in probabilistic terms. There are an infinite number of possible statistical tests, each assessing the presence or absence of a "pattern" which, if detected, would indicate that the sequence is nonrandom. Because there are so many tests for judging whether a sequence is random or not, no specific finite set of tests is deemed "complete." In addition, the results of statistical testing must be interpreted with some care and caution to avoid incorrect conclusions about a specific generator.

TABLE III: PSEUDORANDOM TEST RESULTS OF ALGORITHM 1

Statistical Test	Block / Template length	P-value	Lowest success ratio
Frequency	-	0.816537	99/100
Frequency within blocks	128	0.137282	100/100
Cumulative sums (forward)	-	0.289667	99/100
Cumulative sums (reverse)	-	0.816537	99/100
Runs	-	0.739918	99/100
Longest run within blocks	-	0.574903	100/100
Binary rank	-	0.191687	98/100
FFT	-	0.474986	99/100
NonOverlappingtemplates	9	0.275709	100/100
Overlapping templates	9	0.455937	100/100
Universal	−	0.474986	99/100
Approximate entropy	10	0.595549	100/100
Random excursions	X=-4	0.115387	99/100
	X=-3	0.025193	99/100
	X=-2	0.759756	100/100
	X=-1	0.779188	100/100
	X=1	0.911413	100/100
	X=2	0.350485	100/100
	X=3	0.319084	99/100
	X=4	0.798139	100/100
Random excursions variant (state x)	X=-9	0.195163	68/68
	X=-8	0.043745	68/68
	X=-7	0.005490	68/68
	X=-6	0.275709	68/68
	X=-5	0.002316	68/68
	X=-4	0.066882	68/68
	X=-3	0.275709	68/68
	X=-2	0.671779	68/68
	X=-1	0.671779	68/68
	X=1	0.000839	67/68
	X=2	0.500934	68/68
	X=3	0.500934	68/68
	X=4	0.020085	67/68
	X=5	0.134686	67/68
	X=6	0.911413	67/68
	X=7	0.275709	67/68
	X=8	0.407091	68/68
	X=9	0.100508	68/68
Serial	16	0.699313	98/100
	16	0.350485	99/100
Linear complexity	500	0.616305	98/100

A statistical test is formulated to test a specific null hypothesis (H0). For the purpose of this document, the null hypothesis under test is that the sequence being tested is random.

Each test is based on a calculated test statistic value, which is a function of the data. If the test statistic value is S and the critical value is t, then the Type I error probability is $P(S > t \,||\, Ho$ is true) = P(reject Ho $|0$ is true), and the Type II error probability is $P(S \leq t \,||\, H0$ is false) = P(accept $H0 \mid H0$ is false). The test statistic is used to calculate a *P-value* that summarizes the strength of the evidence against the null hypothesis. For these tests, each *P-value* is the probability that a perfect random number generator would have produced a sequence less random than the sequence that was tested, given the kind of non-randomness assessed by the test. If a *P-value* for a test is determined to be equal to 1, then the sequence appears to have perfect randomness. A *P-value* of zero indicates that the sequence appears to be completely non-random. A significance level (α) can be chosen for the tests. If *P-value* ≥ α, then the null hypothesis is accepted; i.e., the sequence appears to be random. If *P-value* < α, then the null hypothesis is rejected; i.e., the sequence appears to be non-random. The parameter α denotes the probability of the Type I error. Typically, α is chosen in the range [0.001, 0.01].

According to Algorithm 1, we generate 150 Mb of data file. This file can be independently examined by the NIST Statistical Test Suite, and test results are shown in Table III [12].

The minimum pass rate for each statistical test with the exception of the random excursion (variant) test is approximately = 96 for a sample size = 100 binary sequences.

The minimum pass rate for the random excursion (variant) test is approximately = 64 for a sample size = 68 binary sequences.

As seen all the random test outputs are in the dedicated interval and therefore, we can conclude that the data file sequence is random.

V. CONCLUSION

In this paper we discussed about an algorithm for a pseudorandom generation number, which is based on the KECCAK hash function and can pass random test of the NIST Statistical Test Suite and ENT random test. The experimental results show the KECCAK hash function has excellent pseudo randomness, and its security shows that it can be utilized to generate pseudorandom bit sequences.

REFERENCES

[1] S. Pearson and B. Balacheff, "Trusted computing platforms: TCPA technology in context," "*Prentice Hall PTR,"* 2003.

[2] A. Francillon and C. Castelluccia, "TinyRNG: A Cryptographic Random Number Generator for Wireless Sensors Network Nodes," Modeling and Optimization in Mobile, *Ad Hoc and Wireless Networks and Worksho*ps," 2007.

[3] R. A, J. Soto, and J. Nechvatal, A statistical test suit for random pseudorandom number generator for cryptogra-phic applications [EB/OL]. [2009-10-12]. http://csrc.nist.gov/rng/SP800-22b.pdf. Online in valuable.

[4] X. Y. Wang, Y. L .Yin, and H. Yu "Finding collisions in the full SHA-1," http://Proceeding of CRYPTO 2005. Berlin: Springer-Verlag: 17-36. 2005. Online in valuable.

[5] X. Wang, D. Feng, and X. Lai "Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD," http: //Proceeding of Crypto2005. Berlin: Springer-Verlag, 2004: 54-67. Online in valuable.
[6] Z. Yuan, W. Wang, and K. T. Jia, "New birthday attacks on some MACs based on block ciphers," http://Proceedings of CRYPTO 2009. Berlin: Springer-Verlag, 2009: 209-230. Online in valuable.
[7] National Institute of Standards and Technology (NIST). http://csrc.nist.gov. Online in valuable.
[8] G. Bertoni, J. Daemen, M. Peters, and G. Van Assche. KECCAK specifications.http://KECCAK.noekeon.org/KECCAK-specifications. pdf. Online in valuable.
[9] G. Bertoni, J. Daemen, M. Peters, and G. Van Assche. "KECCAK sponge function family main document," http://KECCAK.noekeon.org/KECCAK-main-2.1.pdf. Online in valuable.
[10] F. Buccafurri and G. Lax "A Lightweight Authentication Protocol for Web Applications in Mobile Environments," "*Advanced Information and Knowledge Processing,"* 371-391, DOI: 10.1007/978-1-84996-074-8_14.2010.
[11] FIPS 186, "Digital Signature Standard," *"National Institute of Standards and Technologies*,*"* 1994.
[12] A. Rukbin, J. Soto, and J. Nechvatal, A statistical test suit for random pseudorandom number generator for cryptogra-phic applications [EB/OL]. [2009-10-12]. http://csrc.nist.gov/rng/SP800-22b.pdf. Online in valuable.