Journal of Computer Science 5 (12): 962-973, 2009
ISSN 1549-3636

Omega Network Hash Construction

Chai Wen Chuah and Azman Samsudin
School of Computer Sciences, University Sains Malaysia, Pulau Pinang, Malaysia

Abstract: Problem statement: Cryptographic hash functions are important cryptographic primitives. They are commonly used for data integrity checking and data authentication. Most of the cryptographic hash functions are based on the Merkle-Damgård construction. The basic Merkle-Damgård construction runs over the input sequentially, which can lead to problems when the input size is large since the computation time increases linearly. **Approach:** Therefore, an alternative architecture which can reduce the computation time is needed, especially in today's world where multi-core processors and multithreaded programming are common. An Omega Network Hash Construction (ONHC) run parallel in multi-core machine has been proposed as an alternative to the existing hash constructions. **Result:** The ONHC performed better than the Merkle-Damgård construction. ONHC permutation architecture also showed improved security strength in term of digest value randomness when compared to Merkle-Damgård construction. **Conclusion:** Therefore, it is believed that the proposed ONHC is a valuable structure that can improve the performance of any hash function that can run on top of it.

Key words: Hash function, Merkle-Damgård construction, Omega network, Secure Hash Function (SHA)

INTRODUCTION

Cryptography is becoming more and more important for ensuring various types of security over insecure connections. Among data security primitives, data integrity check and data origin authentication are the common security services that must be applied in many electronic applications, such as electronic commerce, electronic financial transactions, software distribution, electronic mail, data storage and others. Data integrity check is accomplished through the use of cryptographic hash functions, which operate at the root of many other cryptographic methods in achieving these security services.

The basic operation of a hash function is to transform a variable-size input or message into a fixed-length string called a "hash value" or "message digest." A hash value is generated by a function H of the form H(M) = n, where n is the hash value and M is the variable-length input or message. Hash functions are one-way functions; it is easy to generate the digest n from a given message M, but given only n, it is computationally infeasible to generate M. Hash functions are designed to produce unambiguous and condense message digests that are uniquely identifiable with their source messages. However, the source messages cannot be deduced from the message digests and for this reason, the hash function is sometimes known as a digital fingerprint.

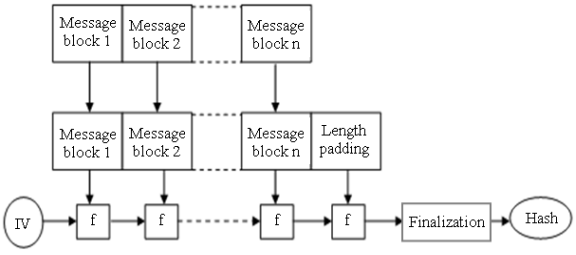


Fig. 1: Merkle-Damgård Construction [adopted from 2, 3, 5]

The architecture of most hash functions is based on the Merkle-Damgård construction[2,3,5,8] (Fig. 1), which is sequential in nature. This means that when the size of the input increases, the computational time will increase linearly. Each step in the Merkle-Damgård construction processes a message block and returns a vector. The first vector is pre-defined, but the remaining vectors are fully dependent on the previous function's output, which slows down the runtime. This in turn has a major effect on the performance of a hash function, SHA, for example. Therefore, there is a need to enhance the performance and efficiency of hashing.

This research proposes the omega network hash construction as an alternative hashing architecture to the Merkle-Damgård construction. Because of the

Corresponding Author: Chai Wen Chuah, School of Computer Sciences, University Sains Malaysia, Pulau Pinang, Malaysia
Tel: (+604) 6533635 Fax: (+604) 6573335

design of the omega network hash construction, the original inputs cannot be retrieved from their corresponding hash values. In the era of multithreading and multi-core technology, the omega network hash construction runs in parallel to improve the hashing performance. The goal of this architecture is to improve performance without sacrificing the security provided by the existing Merkle-Damgård architecture.

Related work: National Institute of Standards and Technology (NIST) organized a competition for selecting SHA-3 currently (2009). ESSENCE[7] is a candidate for this competition. ESSENCE is a cryptographic hashing algorithm from which the construction for hashing algorithm is based on Tree Based Merkle-Damgård construction. The ESSENCE design has been optimized using the parallel implementation and obtained better performance than the sequential Merkle-Damgård construction, but it had a poor performance on short messages.

Pipeline is one of the methods for function decomposition in the field of parallel. Pongyupinpanich and Choomchuay utilized the pipeline method to runs SHA-1, to enhance the performance of Digital Signature Algorithm (DSA) by overcoming unreasonable overhead in small applications[11]. There is small modification done on SHA-1, from which SHA-1 is coded to run in a pipeline mode. $E + W_t + K_t$ is computed parallel with A + B + C + D. The authors claimed that the major drawback from this pipeline design is when there exist a higher number of pipeline states in which the design is cumbersome and the gate count increases dramatically. Thus, this pipeline SHA-1 with DSA is not scalable for all size application because it can only perform well in small size applications.

The Secure Hash Dynamic Structure Algorithm (SHDSA)[2] is used in many applications such as public key cryptosystems, digital signature, digital encryption, message authentication code and random number generators. All of these application's requirements are different from each other. As a result, Elkamchouchi's group proposed SHDSA which comes in a variety of configurations. This dynamic scheme is based on SHA but with one major difference-the hash value is variable length with possible sizes of 128, 192 and 256 bits. Besides that, the iterations in each function can be changeable based on the requirement of the applications. Thus, this dynamic scheme provides different levels of security for satisfying the choices for those practical applications. Although SHDSA is designed to be changeable based on the requirement of the application, the architecture is formulated sequentially and the functions for SHDSA also executed sequentially. The performance is affected; the execution time will increase linearly and reach the highest degree of throughput when the size of input is high. The high speed requirement of SHDSA is highly needed, which is why SHDSA should be parallelized.

Once again, in 2008, Elkamchouchi *et. al*.[3] proposed another secure and fast algorithm called SFHA-256. This one was specifically designed for SHA-256. It is based on the 3C construction, which is based on the Merkle-Damgård construction. The author claims that the proposed architecture is more secure and performs better than the existing SHA-256. He claims that performance is better because the number of operations performed in a step function is reduced and because the architecture consists of two branches running in parallel. SFHA-256 has fewer processing steps, but it is still secure because each step function contains operations that make it difficult for attackers to analyze SFHA-256. The added operations are simple XOR, addition and shift rotation operations. However, performance still suffers due to the waiting time that occurs during the processing of hash values.

Gauravaram *et al*.[5] proposed the 3C+ hash construction which is based on the Merkle-Damgård construction. This 3C+ construction is an enhancement of 3C construction where a third internal chain has been added on top of the cascade and accumulation chains of 3C. With this enhancement, the security level of 3C+ is better than both 3C construction and the Merkle-Damgård construction. 3C+ contains more XOR operations which also improves its security. However, in this new algorithm, there exist conditions where the hashing functions are required to wait for the input from the previous hash function. Moreover, the whole construction is sequential. Thus, waiting times can be extremely high in the 3C+ construction.

Mirvaziri *et al*.[8] came up with an enhancement to the Merkle-Damgård construction by developing a single-length compression function implemented on the Miyaguchi-Preneel block cipher. The architecture has intelligent repetition optimize hash process, which leads to better security. Though the architecture is designed in double levels, it runs sequentially across the message, which means its computation time increases linearly when the input size increases.

In conclusion, most of the proposed architectures run sequentially, which means the computation time increases linearly when the input size increases. Given that multi-core technology and multithreaded programming are common in today's world, these architectures are unacceptably slow.

MATERIALS AND METHODS

Five different sizes of omega network are designed-omega network hash construction 8 (Fig. 2), omega network hash construction 16 (Fig. 3), omega network hash construction 32 (Fig. 3), omega network hash construction 64 (Fig. 3) and omega network hash construction 128 (Fig. 7). They serve as prototypes to determine the optimum size that gives us the best performance when the hash constructs are simulated on dual-core and quad-core processors machines.

As a prototype, SHA-512 algorithm is used as the function for Omega Network Hash Construction. Merkle-Damgård construction run sequentially, SHA-512 algorithm needs only one set of constant values (80 constant 64-bit words which are parts of the cube roots of the first eighty prime numbers[9]). In Omega Network Hash Construction, the constant values are taken from part of the square root of 2 ($\sqrt{2}$) and the number of constant value used depends on the number blocks (Table 1).

In the design of Omega Network Hash Construction (Fig. 2 and 3), the blocks on the left column (Column 1) takes two categories of input-1024 and 512 bits. Both inputs are taken from user messages. Those inputs serve as the initial vector for the SHA 512 function. However, in Merkle-Damgård construction, the initial vector is predefined. Each block of column 2 and above takes the input vector from the XOR of two blocks of digested values from previous column. For example (Fig. 2), block function number F_4 gets its input vector value from the XOR of two digest values of block F_0 and F_2.

There are some constraints for the design of Omega Network Hash Construction. To start executing the block function's column, it must wait for the previous column to complete its execution before it starts. This is because the block function's column input vector depends on the XOR of digest values of the previous column; so variables dependency does exist in Omega Network Hash Construction. For example, second column of Omega Network Hash Construction can start execute only after the first column completed its execution, because the input vector of second column depends on the XOR of first column digested values.

One round of Omega Network Hash Construction is completed, after XORing the digest values of the last column. This process continues until the entire message is hashed. Finally, to form the final digest value, digest values of every round is XORed. The XOR process is executed sequentially, so that when the size of Omega Network Hash Construction increases, this process will take longer time to XOR the digest value.

If the input message is not enough in terms of the input length to complete one round of Omega Network Hash Construction, the pre-defined message will be used to execute the remaining blocks. The pre-defined message is a non-repeated number, taken from the irrational number $\sqrt{2}$.

Table 1: Set of constant value for omega network hash construction

Size of omega network hash construction	8	16	32	64	128
Set of constant value	12	32	80	192	448

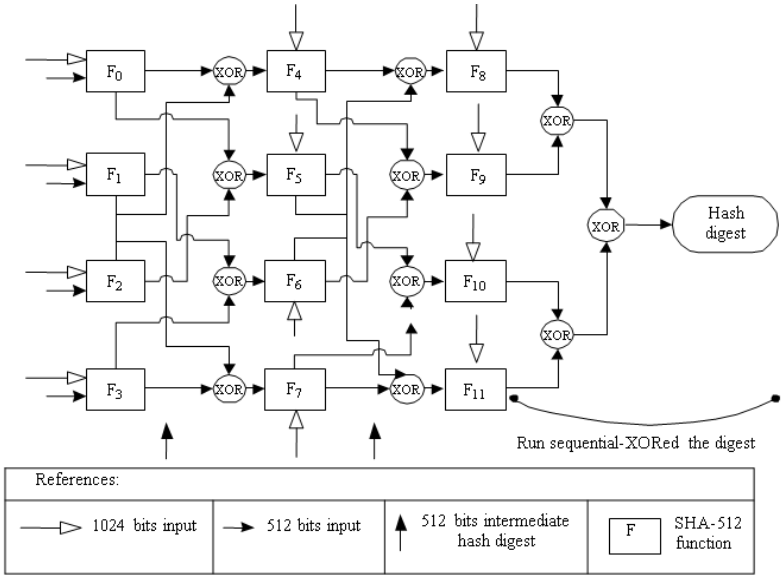
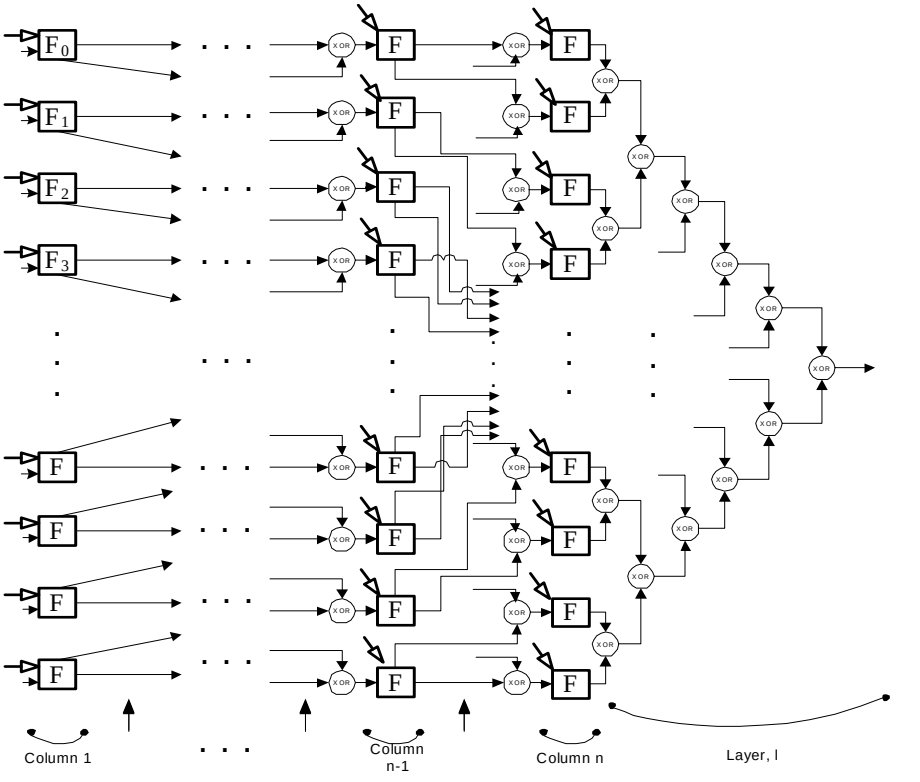


Fig. 2: Omega network hash construction 8



Size of Omega Network Hash Construction	Total Blocks (Hash Function)	Total Columns (n)	Total Layer (l)
16	32	4	3
32	80	5	4
64	192	6	5
128	448	7	6

References:

1024 bits input	512 bits input
512 bits intermediate hash digest	F SHA-512 Function

Fig. 3: Overview of 4 types of omega network hash construction

Overall, the executions phase of the different versions of omega network hash construction are similar to each other. The difference is the number of columns and the number of function blocks that form an omega network hash construction. Besides that, the total rounds needed to complete hashing a message are different for these five designs of omega network hash construction.

Simulation and performance evaluation: All designs are simulated in dual core and quad core machines. Section on test design describes the complete test design for this research. Section on performance test explains how to generate the test file and the sizes of test final. Section on security test describes the security test for this research.

Test design: Omega network hash construction is proposed to be an alternative architecture to the Merkle-Damgård construction with the goal of improving the performance of hashing. SHA-512 algorithm is used as our case study in this research. The source code of the SHA-512 function is taken from Olivier Gay version 2007[10].

In our test, there are five major designs of omega network hash construction, with different sizes, being simulated on both dual-core processors and quad-core processors. To execute omega network hash construction in parallel, two lines of OpenMP commands are used.

- omp_get_num_procs (): This command is used to get the number of processors in the machine. The command will help us to indicate the number of threads created. Two threads will be created in a dual-core processor and four threads will be created for a quad-core processor, respectively

- #pragma omp parallel for schedule (dynamic) ordered private (iCount): This OpenMP command provides the Omega Network Hash Construction's function (SHA-512) to execute parallel schedules orderly from F_1, F_2, F_3, F_4, F_5, F_6, F_7, F_8…F_n, each thread will execute one function at a time in parallel with the other thread. iCount is a private variable for all the functions

The designs are simulated in two types of machine which both are using-Microsoft Windows XP Professional and Visual Studio 2008. The specifications of the machines are as below:

- Dual-core processors:
  Intel (R) core (TM) 2 duo CPU T7300 @ 2.00 GHz
  1.99 GHz 2.00 GB of RAM
  Physical address extension
  Hard disk: 80 GB
- Quad-core processors
  AMD phenom™ 9650 quad-core processor
  2.30 GHz 3.00 GB of RAM
  Physical address extension
  Hard disk: 230 GB

Performance test: The performance testing for five types of omega network are done by executing them in two types of machine as mentioned above. A total of 5 files had been used for testing which are 200, 400, 600 and 800 MB and 1 GB. All types of omega network hash construction are executed five times for each file; the average execution times are recorded.

Security test: To evaluate the security strength of Omega Network Hash Construction, we use DIEHARD random test[6]. DIEHARD, the security test needs 9-10 megabytes of files containing the binary value to produce the test report.

There are two types of messages digest value being generated-single block message (Fig. 4) and multiple blocks message (Fig. 5). Each input file is only one bit different for both types of messages. A total of 1,600,000 message files are generated to produce 10 MB binary digested value file. For the single block message, the first byte is "1", followed by 127 "0" (Fig. 4). There are 5 types of messages for multiple blocks-12, 32, 80, 192 and 448 blocks. The number bits of message for Omega Network Hash Constructions are greater than Merkle-Damgård construction because the initial vector for Omega Network Hash Construction is taken from user input (Fig. 5). The test results are written in an output file. The test value '-p' between 0 and 1 indicates a pass for the DIEHARD test.

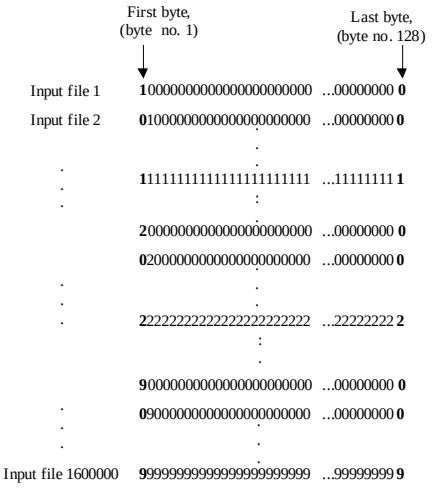


Fig. 4: Test data, single block of message

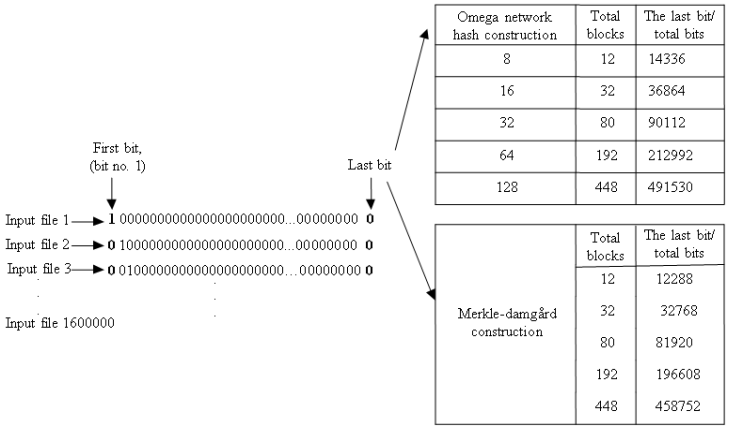


Fig. 5: Test data, multiple blocks of message

RESULTS

Performance test: The performance analysis measures speed up, security level, efficiency and running cost. There are three types of run-time tests: serial run-time (T_s), parallel run-time (T_p) and overhead function (T_0). T_s is the time elapsed between the beginning and the end of execution time in a serial manner. T_p is the elapse time between the moment a parallel computation starts and the moment that the last processor finishes execution. Overhead is calculated by $T_o = p\ T_p - T_s$, where p is the number of threads.

Speed up is defined as the execution time of a sequential program divided by the execution time of a parallel program, $S = T_s/T_p$, where T_s is the sequential time and T_p is the parallel time running on n processors[1]. Theoretically, speed up will not exceed the number of processors or threads that is being used.

The efficiency is calculated as E = Speed up/n. This measurement is the fraction of time for which the computer is employed. Ideally, the efficiency should equal to one. But in practice, the efficiency is normally between zero and one, which depends on the degree of effectiveness the processors are being utilized.

Running cost is another measurement we considered in our performance analysis. Cost or ($p \times T_p$) is the product of parallel runtime and the number of processing elements which reflects the sum of the time that each processing element spends to solve a problem.

Performance of omega network hash construction 8, omega network hash construction 16, omega network hash construction 32, omega network hash construction 64 and omega network hash construction 128 are shown in Table 2-6 respectively. Figure 6-15 shows performance analysis of Omega Network Hash Construction simulated on a dual core machine and quad core machine in the line graph format.

Table 2: Performance analysis of omega network hash construction 8

Omega network hash construction 8

Sizes	2 Threads						4 Threads					
(MB)	T_s (sec)	T_p (sec)	T_o	S	E	Running cost	T_s (sec)	T_p (sec)	T_o	S	E	Running cost
200	16.188	11.416	6.643	1.418	0.355	22.831	14.734	10.844	28.642	1.359	0.340	43.376
400	32.375	23.742	15.110	1.364	0.341	47.485	29.407	21.875	58.093	1.344	0.336	87.500
600	48.547	35.080	21.613	1.384	0.346	70.160	44.109	32.719	86.767	1.348	0.337	130.876
800	64.778	46.538	28.297	1.392	0.348	93.075	58.891	42.375	110.609	1.390	0.347	169.500
1000	80.942	58.542	36.143	1.383	0.346	117.085	73.625	56.922	154.063	1.293	0.323	227.688

Table 3: Performance analysis of omega network hash construction 16

Omega network hash construction 16

Sizes	2 Threads						4 Threads					
(MB)	T_s (sec)	T_p (sec)	T_o	S	E	Running cost	T_s (sec)	T_p (sec)	T_o	S	E	Running cost
200	16.188	12.064	7.940	1.342	0.335	24.128	14.734	10.219	26.142	1.442	0.360	40.876
400	32.375	24.177	15.978	1.339	0.335	48.353	29.407	20.687	53.341	1.422	0.355	82.748
600	48.547	35.978	23.410	1.349	0.337	71.957	44.109	31.062	80.139	1.420	0.355	124.248
800	64.778	47.890	31.002	1.353	0.338	95.780	58.891	41.765	108.169	1.410	0.353	167.06
1000	80.942	59.878	38.813	1.352	0.338	119.755	73.625	53.594	140.751	1.374	0.343	214.376

Table 4: Performance analysis of omega network hash construction 32

Omega network hash construction 32

Sizes	2 Threads						4 Threads					
(MB)	T_s (sec)	T_p (sec)	T_o	S	E	Running cost	T_s (sec)	T_p (sec)	T_o	S	E	Running cost
200	16.188	12.485	8.782	1.297	0.324	24.970	14.734	8.281	18.390	1.779	0.445	33.124
400	32.375	24.910	17.445	1.300	0.325	49.820	29.407	16.219	35.469	1.813	0.453	64.876
600	48.547	37.232	25.917	1.304	0.326	74.464	44.109	24.516	53.955	1.799	0.450	98.064
800	64.778	49.563	34.348	1.307	0.327	99.126	58.891	32.516	71.173	1.811	0.453	130.064
1000	80.942	61.922	42.902	1.307	0.327	123.844	73.625	41.031	90.499	1.794	0.449	164.124

Table 5: Performance analysis of omega network hash construction 64

Omega network hash construction 64												
	2 Threads						4 Threads					
Sizes (MB)	T_s (sec)	T_p (sec)	T_o	S	E	Running cost	T_s (sec)	T_p (sec)	T_o	S	E	Running cost
200	16.188	12.516	8.844	1.293	0.323	25.032	14.734	9.813	24.518	1.501	0.375	39.252
400	32.375	25.141	17.907	1.288	0.322	50.282	29.407	19.719	49.469	1.491	0.373	78.876
600	48.547	37.532	26.517	1.293	0.323	75.064	44.109	29.469	73.767	1.497	0.374	117.876
800	64.778	50.641	36.504	1.279	0.320	101.282	58.891	39.734	100.045	1.482	0.371	158.936
1000	80.942	63.219	45.496	1.280	0.320	126.438	73.625	49.469	124.251	1.488	0.372	197.876

Table 6: Performance analysis of omega network hash construction 128

Omega network hash construction 128												
	2 Threads						4 Threads					
Sizes (MB)	T_s (sec)	T_p (sec)	T_o	S	E	Running cost	T_s (sec)	T_p (sec)	T_o	S	E	Running cost
200	16.188	13.047	9.906	1.241	0.310	26.094	14.734	22.563	75.518	0.653	0.163	90.252
400	32.375	26.125	19.875	1.239	0.310	52.250	29.407	45.422	152.281	0.647	0.162	181.688
600	48.547	39.032	29.517	1.244	0.311	78.064	44.109	68.718	230.763	0.642	0.160	274.872
800	64.778	52.078	39.378	1.244	0.311	104.156	58.891	91.313	306.361	0.645	0.161	365.252
1000	80.942	65.203	49.464	1.241	0.310	130.406	73.625	108.437	360.123	0.679	0.170	433.748

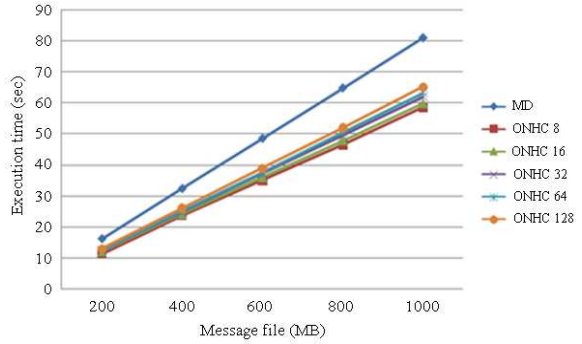


Fig. 6: Execution time comparison between omega network hash construction and Merkle-Damgård Construction, simulated on a dual core machine

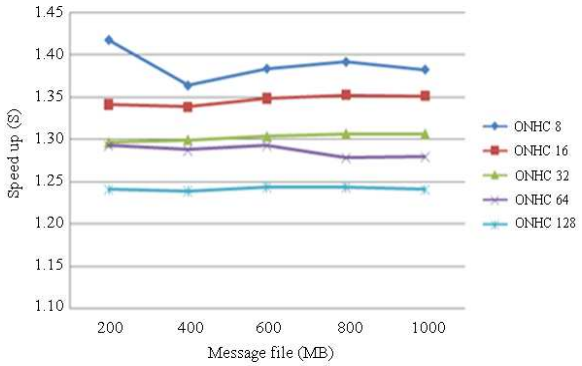


Fig. 8: Speed up of omega network hash construction simulated on a dual core machine

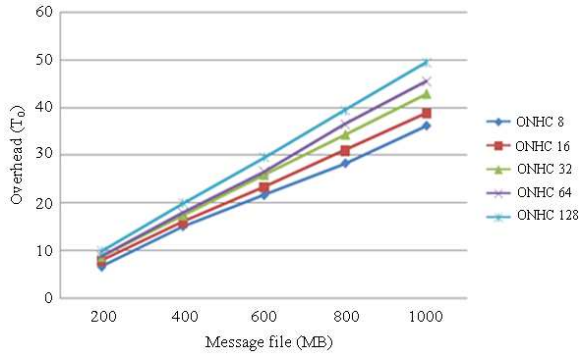


Fig. 7: Overhead of omega network hash construction simulated on dual core machine

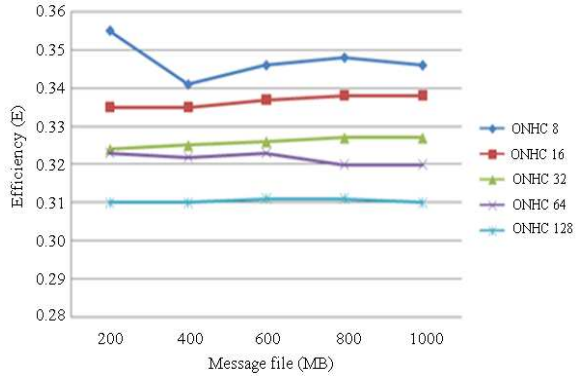


Fig. 9: Efficiency of omega network hash construction simulated on a dual core machine

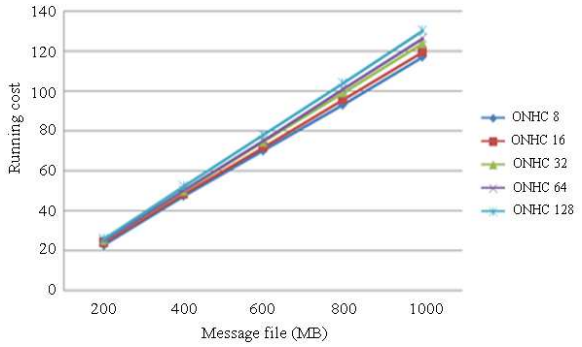


Fig. 10: Running cost of omega network hash construction simulated on a dual core machine

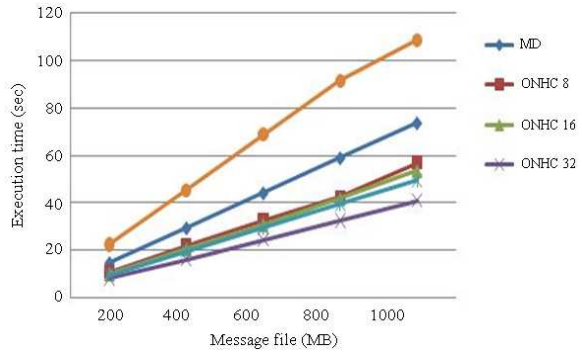


Fig. 11: Execution time comparison between omega network hash construction and Merkle-Damgård construction, simulated on a quad core machine

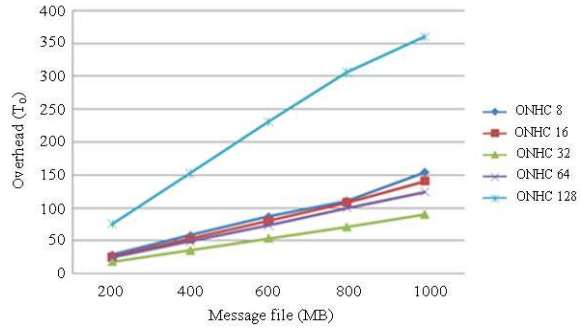


Fig. 12: Overhead of omega network hash construction simulated on a quad core machine

Security analysis: DIEHARD is chosen as the security test tool to examine whether the omega network hash constructions are secure enough by evaluating the randomness of the digest value. There are two types of input file, single block of message and multiple blocks of message. Each input file is only one bit different for both types of message. The result of security analysis for single block message is shown in Table 7 and 8 shows the security analysis for multiple block messages.

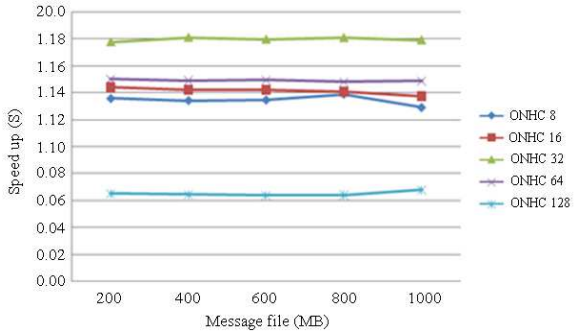


Fig. 13: Speed up of omega network hash construction simulated on a quad core machine

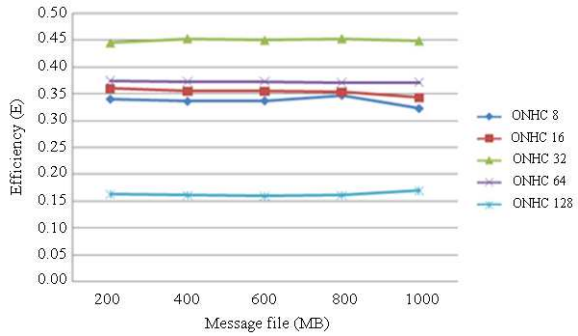


Fig. 14: Efficiency of omega network hash construction simulated on a quad core machine

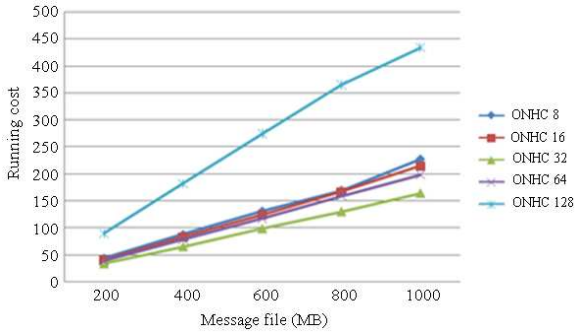


Fig. 15: Running cost of omega network hash construction simulated on a quad core machine

Table 7: DIEHARD security test, single block of message to generate binary test file

Single block of message	MD	ONHC 8	ONHC 16	ONHC 32	ONHC 64	ONHC 128
1. Birthday spacings	Fail*	0.6484800***	0.9209820***	0.343668***	0.7568980***	0.627634***
2. Overlapping permutations	1.000000**	0.1214495***	0.5986095***	0.506419***	0.8482345***	0.303512***
3. Ranks of 31×31 and 32×32 matrices						
31×31 matrices	1.000000**	0.6794880***	0.525663***	0.360214***	0.6628150***	0.772463***
32×32 matrices	1.000000**	0.7980960***	0.642444***	0.521738***	0.3315500***	0.380405***
4. Ranks of 6×8 matrices	1.000000**	0.6799860***	0.201000***	0.594762***	0.1440980***	0.623201***
5. Monkey tests on 20 bit words	1.000000**	0.5627780***	0.566308***	0.379957***	0.4361720***	0.411402***
6. Monkey tests OPSO, OQSO, DNA						
OPSO	1.000000**	0.5375000***	0.589404***	0.621691***	0.5285300***	0.560243***
OQSO	1.000000**	0.5275430***	0.514518***	0.431471***	0.4967710***	0.545821***
DNA	1.000000**	0.5299740***	0.512017***	0.559690***	0.5366310***	0.566177***
7. Count the 1's in a stream of bytes	1.000000**	0.5343885***	0.652317***	0.638018***	0.3421095***	0.594358***
8. Count the 1's in specific bytes	1.000000**	0.5106000***	0.584141***	0.520437***	0.6004920***	0.544833***
9. Parking lot test	1.000000**	0.6257840***	0.263863***	0.610674***	0.5914920***	0.504483***
10. Minimum distance test	1.000000**	0.9708900***	0.957626***	0.087640***	0.6157280***	0.904798***
11. Random spheres test	1.000000**	0.9898730***	0.018399***	0.980629***	0.9839010***	0.900696***
12. The squeeze test	1.000000**	0.6085880***	0.104880***	0.486552***	0.8308000***	0.919765***
13. Overlapping sums test	1.000000**	0.4132320***	0.836154***	0.487671***	0.7555970***	0.694437***
14. Runs test	1.000000**	0.2870380***	0.541360***	0.743451***	0.5838500***	0.514282***
15. The craps test	0.04061***	0.1290400***	0.048140***	0.085660***	0.8882300***	0.172820***

Note: MD: Merkle-Damgård construction; ONHC: Omega Network Hash Construction; *: Fail to test; ***: Pass the DIEHARD test; **: Fail the DIEHARD test

Table 8: DIEHARD security test, multiple blocks of message to generate binary test file

Multiple blocks of message	MD -8 blocks	ONHC 8	MD-16 blocks	ONHC 16	MD-32 blocks	ONHC 32	MD-64 blocks	ONHC 64	MD-128 blocks	ONHC 128
1. Birthday spacings	Fail*	0.746930***	0.999999***	0.745669***	0.9251170***	0.654962***	0.918840***	0.296097***	0.8663900***	0.081941***
2. Overlapping permutations	0.991743***	0.403184***	0.686777***	0.160407***	0.1904655***	0.274975***	0.831856***	0.761285***	0.2887360***	0.839987***
3. Ranks of 31×31 and 32×32 matrices										
31×31 Matrices	1.000000**	0.378031***	1.000000**	0.580842***	0.3808820***	0.323987***	0.613335***	0.439781***	0.6478570***	0.700023***
32×32 Matrices	1.000000**	0.342838***	1.000000**	0.431458***	0.3309560***	0.997820***	0.507316***	0.320859***	0.8975000***	0.384348***
4. Ranks of 6×8 Matrices	0.994176***	0.663811***	0.487267***	0.596087***	0.8598710***	0.321908***	0.913995***	0.696835***	0.4728510***	0.809089***
5. Monkey tests on 20 bit words	0.972868***	0.469094***	0.911032***	0.515542***	0.5202000***	0.605502***	0.440477***	0.562937***	0.5420770***	0.540819***
6. Monkey Tests OPSO, OQSO, DNA										
OPSO	1.000000**	0.651809***	1.000000**	0.440704***	0.6097480***	0.567804***	0.446613***	0.517261***	0.5947910***	0.535348***
OQSO	1.000000**	0.476664***	1.000000**	0.508654***	0.4990460***	0.360325***	0.460171***	0.572464***	0.4527750***	0.486843***
DNA	1.000000**	0.410194***	1.000000**	0.462739***	0.5689680***	0.526210***	0.517032***	0.411974***	0.4490290***	0.564490***
7. Count the 1's in a stream of bytes	1.000000**	0.381799***	0.998289***	0.736791***	0.0668032***	0.754137***	0.506117***	0.490607***	0.6764915***	0.851321***
8. Count the 1's in specific bytes	0.709227***	0.570537***	0.683469***	0.545262***	0.5214650***	0.495342***	0.475061***	0.514982***	0.5365590***	0.671321***
9. Parking lot test	0.996457***	0.232971***	0.513322***	0.992594***	0.6289940***	0.863783***	0.621359***	0.378580***	0.4936960***	0.794248***
10. Minimum distance test	1.000000**	0.980371***	1.000000**	0.515071***	1.0000000**	0.985678***	0.961815***	0.717499***	0.9486340***	0.080092***
11. Random spheres test	0.994149***	0.625952***	0.414147***	0.121640***	0.9591750***	0.479698***	0.635821***	0.622163***	0.5002240***	0.712335***
12. The squeeze test	1.000000**	0.420809***	1.000000**	0.217036***	0.0604030***	0.907871***	0.104538***	0.165316***	0.3754680***	0.737879***
13. Overlapping sums test	0.797626***	0.073680***	0.321314***	0.673886***	0.9730550***	0.939852***	0.466093***	0.536601***	0.5133860***	0.027135***
14. Runs test	0.578203***	0.359220***	0.306128***	0.436846***	0.3360040***	0.500120***	0.441318***	0.410291***	0.5390950***	0.494026***
15. The craps test	0.042190***	0.551850***	0.185690***	0.651640***	0.3910200***	0.379050***	0.528770***	0.927670***	0.9575200***	0.621430***

Note: MD: Merkle-Damgård construction; ONHC: Omega Network Hash Construction; *: Fail to test; ***: Pass the DIEHARD test; **: Fail the DIEHARD test

DISCUSSION

Performance analysis: All sizes of omega network hash construction and Merkle-Damgård construction are simulated on dual core and quad core machines respectively. The features of each machines are describe in the section of Test Design. The amount of RAM of the quad core machine (2.30 GHz 3.00 GB of RAM) is higher than dual core machine (1.99 GHz 2.00 GB of RAM). Thus, execution time for omega network hash constructions and Merkle-Damgård construction which are simulated on the quad core machine is faster than the simulation result on dual core machine.

As mentioned earlier, waiting time and serial time are two constraints exist in the design of omega network hash constructions. The waiting time is higher when the size of omega network hash construction is small. (e.g., omega network hash construction 8). The serial time is higher when the size of Omega Network Hash Construction is large (e.g., omega network hash construction 128). On the other hand, when more threads are created, the communication time or overhead will be higher. Thus, communication time for four threads is higher than two threads. Waiting time, serial time, communication time (overhead) are three factors that affect the performance of omega network hash constructions.

On dual core processors, the execution time among all sizes of omega network hash construction are almost the same with each other. The differences in the execution time among the difference sizes are created by digest XORing process happen in the last column of the design which runs sequentially. When the size of Omega Network Hash Construction increases, it requires longer time to complete the XORing process.

Omega network hash construction 8 runs faster on dual core processors, while omega network hash construction 128 runs slowest on dual core processors. The reason is the communication time between these two threads for omega network hash construction 8 is less than the overhead time of omega network hash construction 128. Omega network hash construction 128 required more time to generate the digest value by XORing the digest value from the last column compare with omega network hash construction 8.

On a quad core machine, the program will automatically generate four threads to run four block functions of omega network hash construction simultaneously. Omega network hash construction 32 runs faster on quad core processors, while omega network hash construction 128 runs the slowest on quad core processors. Omega network hash construction 8 and 16 had similar execution time. However, omega network hash construction 16 is slower than omega network hash construction 8 because the serial execution of omega network hash construction 16 for XORing the digest value requires more processing time. However, both constructions are still slower than omega network hash construction 32. This happens due to the waiting time that exists in omega network hash construction 8 and 16 are higher than omega network hash construction 32; the more number of execution round the more waiting time is required.

Omega network hash construction 128 runs the slowest on dual core and quad core machines. However, it still runs faster than Merkle-Damgård construction on dual core processors but run slower than Merkle-Damgård construction on quad core processors. This is because the serial time to XOR the digest values is higher on both machines but the communication time for four threads are higher than two threads.

There are three types of speed up calculation being presented: The speed up calculation based on (T_s/T_p), the speed up based on Amdahl's law and the speed-up based on Gustafson Barsis's law (Table 10). The fastest speed up is achieved by the omega network hash construction 32 with four threads, 1.813 sec, for input file size of 400 MB (Fig. 7 and 12) and the average speed up is 1.8 sec (Table 9). Based on Amdahl's law and Gustafson Barsis's law, the speed up for omega network hash construction 8 is the highest because the serial execution in omega network hash construction 8 is only 10% for the entire execution. The lowest speed up is by omega network hash construction 128 which consists 50% execution (Table 10). Consequently, the efficiency for Omega Network Hash Construction 32 with four threads is also among the highest (Fig. 8 and 13, Table 11 and 12).

Table 9: Comparison of speed up among the omega network hash constructions

ONHC										
	2 Threads					4 Threads				
Sizes (MB)	8	16	32	64	128	8	16	32	64	128
200	1.418	1.342	1.297	1.293	1.241	1.359	1.442	1.779	1.501	0.653
400	1.364	1.339	1.300	1.288	1.239	1.344	1.422	1.813	1.491	0.647
600	1.384	1.349	1.304	1.293	1.244	1.348	1.420	1.799	1.497	0.642
800	1.392	1.353	1.307	1.279	1.244	1.390	1.410	1.811	1.482	0.645
1000	1.383	1.352	1.307	1.280	1.241	1.293	1.374	1.794	1.488	0.679
Average	1.388	1.347	1.303	1.287	1.242	1.347	1.414	1.799	1.492	0.653

Table 10: Comparison of speed up among the omega network hash constructions based on Amdahl's law and Gustafson Barsis's law

			Average speed up $S = T_s/T_p$		Amdahl's law speed up $S = N/[\beta N+(1-\beta)]$		Gustafson Barsis's law speed up $S = N-(N-1)a$	
ONHC	Serial code	Parallel code	2T	4T	2T	4T	2T	4T
8	0.1	0.9	1.388	1.347	1.818	3.077	1.900	3.700
16	0.2	0.8	1.347	1.414	1.667	2.500	1.800	3.400
32	0.3	0.7	1.303	1.799	1.538	2.105	1.700	3.100
64	0.4	0.6	1.287	1.492	1.428	1.818	1.600	2.800
128	0.5	0.5	1.242	0.653	1.333	1.600	1.500	2.500

Table 11: Comparison of the efficiency among the omega network hash constructions

ONHC										
	2 Threads					4 Threads				
Sizes (MB)	8	16	32	64	128	8	16	32	64	128
200	0.355	0.335	0.324	0.323	0.310	0.340	0.360	0.445	0.375	0.163
400	0.341	0.335	0.325	0.322	0.310	0.336	0.355	0.453	0.373	0.162
600	0.346	0.337	0.326	0.323	0.311	0.337	0.355	0.450	0.374	0.160
800	0.348	0.338	0.327	0.320	0.311	0.347	0.353	0.453	0.371	0.161
1000	0.346	0.338	0.327	0.320	0.310	0.323	0.343	0.449	0.372	0.170
Average	0.347	0.337	0.326	0.322	0.310	0.337	0.353	0.450	0.373	0.163

Table 12: Comparison of the efficiency among the omega network hash constructions based on Amdahl's law and Gustafson Barsis's law

	Efficiency		Efficiency based on Amdahl's law speed up		Efficiency based on Gustafson Barsis's law Speed up	
	$E = (T_s/T_p)/p$		$E = \{ N /[\beta N+(1-\beta)]\}/p$		$E = [N-(N-1)a]/p$	
ONHC	2T	4T	2T	4T	2T	4T
8	0.694	0.336	0.909	0.769	0.950	0.925
16	0.673	0.353	0.833	0.625	0.900	0.850
32	0.651	0.449	0.769	0.526	0.850	0.775
64	0.643	0.373	0.714	0.454	0.800	0.700
128	0.621	0.163	0.666	0.400	0.750	0.625

Running cost is calculated by multiplying the number of threads by the execution time. Therefore, the more the number of processors, the higher the running cost will be recorded for the simulation (Fig. 10 and 15). Thus, the cost of running four threads is higher than that of two threads.

Overall, omega network hash construction 32 runs faster on quad core processors while Omega Network Hash Construction 8 runs faster on dual core processors.

Security analysis: Merkle-Damgård Construction fails one of the security tests-Birthday Spacings. This is because the binary digest value of Merkle-Damgård construction for both cases (single and multiple blocks message) did not fulfill the requirement for Birthday Spacings test. Merkle-Damgård construction generates the lambda equal to 3.000 which is over the maximum lambda setting (2.000) for Birthday Spacings.

For single block of message, all the omega network hash constructions passed the test. On the other hand, Merkle-Damgård construction failed most of the tests except for the craps test (Table 7).

The second test utilizes multiple blocks of message. All the omega network hash construction passed the test, the test value '-p' for each test lie between 0 and 1. Merkle-Damgård construction failed some of the tests when the block size messages are 12 (failed 8 tests), 32 (failed 7 tests) and 80 (failed 1 tests) (Table 8). This implies that, Merkle-Damgård construction can only provide better security in term of randomness when the size of message is higher, because bigger size message will produce more equal size of block message and the digest value will be mixed with each other better and therefore produces the better randomize digest value.

To conclude the security analysis, in term of randomness, for omega network hash construction and with Merkle-Damgård construction, it can be deduced that omega network hash construction produces better randomized digest value even though the message is small and passed basic security test-DIEHARD.

CONCLUSION

The main objective of this research is to design a better hash constructions mechanism that can help improve the hash function performance in general. The propose omega network hash constructions were tested on dual core and quad core processors which allowed the block functions to run in parallel. Waiting time and serial time are two major weaknesses for omega network hash construction which affect its performance. For bigger size omega network hash construction, higher degree of serial execution has been recorded (e.g., omega network hash construction 128). While for the smaller size omega network hash construction, higher degree of waiting time has been recorded (omega network hash construction 8). However, in all cases, omega network hash constructions perform better than Merkle-Damgård construction, especially for the omega network hash construction 32 with four threads, which runs the fastest among all the constructions. In terms of security, all

sizes of omega network hash constructions passed the basic security test of DIEHARD. On the other hand, Merkle-Damgård construction failed most of the security tests when the input size is small. Based on both of the tests, performance test and random test, the omega network hash construction performs better than the well known Merkle-Damgård construction. Therefore Omega Network Hash Construction is a viable alternative for hash construction, especially when multi-core processors are being considered.

REFERENCES

1. Bal, H.E. and M. Haines, 1998. Approaches for integrating task and data parallelism. IEEE Concurr., 6: 78-84. http://portal.acm.org/citation.cfm?id=614113
2. Elkamchouchi, H.M., A.A.M. Einarah and E.A.A. Hagras, 2006. A new Secure Hash Dynamic Structure Algorithm (SHDSA) for public key digital signature schemes. Proceeding of the 23rd Radio Science Conference, Mar. 14-16, IEEE Xplore Press, Menoufiya, pp: 1-9. DOI: 10.1109/NRSC.2006.386347
3. Elkamchouchi, H.M., M.E. Nasr and R.I. Abdelfatah, 2008. A new Secure and Fast Hashing Algorithm (SFHA-256). Proceeding of the 25th National Radio Science Conference, Mar. 18-20, IEEE Xplore Press, Tanta, pp: 1-8. DOI: 10.1109/NRSC.2008.4542348
4. Emam, S.A. and S.S. Emami, 2007. Design and implementation of a fast, combined SHA-512 on FPGA. Int. J. Comput. Sci. Network Secur., 7: 165-168. http://paper.ijcsns.org/07_book/200705/20070524.pdf
5. Gauravaran, P., W. Millan, E. Dawson and K. Viswanathan, 2006. Constructing secure hash functions by enhancing Merkle-Damgård construction. Lecture Notes Comput. Sci., 4058: 407-420. DOI: 10.1007/11780656_34
6. Marsaglia, G., 1996. DIEHARD: A battery of test of randomness. http://i.cs.hku.hk/~diehard/cdrom
7. Martin, J.W., 2009. ESSENCE: A candidate hashing algorithm for the NIST competition. http://www.math.jmu.edu/~martin/essence/Supporting_Documentation/essence_NIST.pdf
8. Mirvaziri, H., K. Jumari, M. Ismail and M. Hanapi, 2007. Collision free hash function based on Miyaguchi-Preneel and enhanced Merkel-Damgard scheme. Proceeding of the 5th Student Conference on Research and Development-SCOReD, Dec. 11-12, IEEE Xplore Press, Malaysia, pp: 1-6. DOI: 10.1109/SCORED.2007.4451411
9. NIST., 1993. Announcing the standard for secure hash standard. http://www.itl.nist.gov/fipspubs/fip180-1.htm
10. Olivar, G., 2007. FIPS 180-2 SHA-224/256/384/512 implementation. http://www.ouah.org/ogay/sha2/
11. Pongyupinpanich, S. and S. Choomchuay, 2004. An architecture for a SHA-1 applied for DSA. Proceedings of the 3rd Asian International Mobile Computing Conference, May 26-28, Thailand, pp: 1-5. http://www.kmitl.ac.th/~kchsomsa/somsak/papers/amoc04-022.pdf