Design of an Intelligent SHA-1 Based Cryptographic System: A CPSO Based Approach

Monjur Alam and Sonai Ray
(Corresponding author: Monjur Alam)

Research & Development Division
Cadence Design Systems (I) Pvt. Ltd., Noida 201305, UP, India
(e-mail:{alammonjur, sonai.ray}@gmail.com)

(Received May 9, 2011; revised and accepted July 13, 2011)

Abstract

The paper puts forward the design of an intelligent SHA-1 based crypto system. For a given 512-bit message stream the intelligence of the system lies in its power of predicting the probable-colluders. Along with the conventional SHA-1 architecture, our scheme employs a predictor control block which takes the message stream from the user, and provides the log-list of the equal length bit-streams that are most likely to produce collisions with the message stream. The predictor controller uses Canonical Particle Swarm optimization (CPSO) algorithm. Besides proposing the algorithm itself, the paper also surveys the performance of the predictor when employed with different hardware platforms.

Keywords: SHA-1, Collision, CPSO

1 Introduction

In recent years, unprecedented advancement in the field of computing power has seen the decline of DES or triple DES [7, 12], as cryptographic algorithms. Also, with the advent of E-commerce, Internet Banking, the concept of digital signature or message authentication [2, 6] has been one of the core issues in the mind of a crypto system designer. Secured Hash Function algorithm-1(SHA-1) [1, 8, 13, 15, 20] is one of the most successful and used hashing function employed till date to provide message authenticity. Compared to Message Digest Algorithm (MD) versions tt has been proved to be less vulnerable towards cryptanalytic attacks.

However despite having satisfied the necessary strong and weak collision [18] properties, the SHA1 based real time crypto systems are not free from collisions. So over the years, the presence of a predictor which can provide the designer with the list of bit patterns which are capable of colliding with the message bit streams has been incumbent in the conventional SHA1 architecture [23]. In a normal SHA-1 the message stream is 512 bit long. So employment of a brutal force method would involve 2^{512} iterations to get the complete list.

In this paper we have explored a hitherto untouched area of SHA-1 and CPSO synergism. Here our proposed predictor scheme employs CPSO based algorithm, which takes fur few steps to give the designer a comprehensive list of the so-called "probable colluders".

Our paper is organized as follows. In Section 2 we discuss the general architecture of SHA1 along with its properties. Section 3 explores CPSO based algorithm. In Section 4 we provide the building blocks and the algorithm of our proposed predictor-attached SHA-1 system. Section 5 contains all the relevant experimental results and a brief overview of our s/w resources. Section 6 concludes our paper.

2 General Architecture of SHA-1 and its Properties

There are three basic modules or blocks in SHA-1 hashing. They are given as below:

- *Padding Block*: The message is initially padded so that its length is congruent to 448 modulo 512. Then a 64-bit representation of the length in bits of the original message is appended to this one, so that the message stream is ultimately 512 bit long.

- *Initialization of MD buffer*: In our case we assigned the buffers with these initial values. $A =$ 67452301, $B = EFCDAB89$, $C = 98BADCFE$, $D =$ 10325476, $E = C3D2E1F0$.

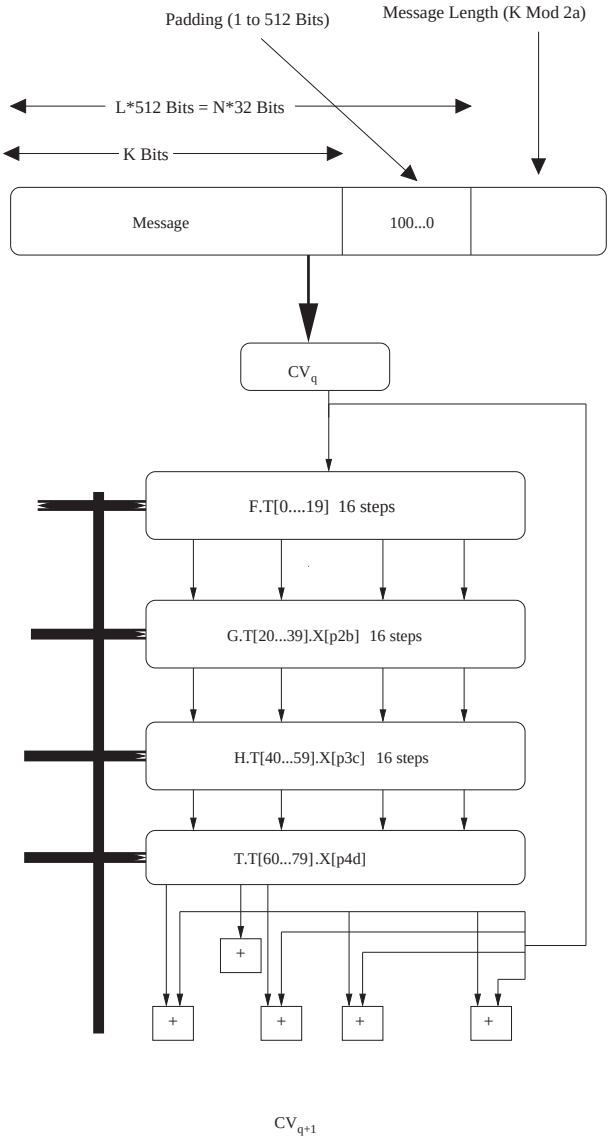

Figure 1: Basic modules of SHA-1

- *Message Processing*: The message is further processed as below:

$$\begin{aligned} CV_{q+1} &= \Sigma\Phi(x) mod 2^{32} \\ \Phi(x) &= F(CV_q, (ABCDE)_q, \end{aligned}$$

where CV_0 = Initialized Buffer value, L = Number of blocks in the message, MD = Final hash value, $(ABCDE)_q$ = o/p of q^{th} message block. A single execution of MD5 involves 20 steps of operations performed on the initial buffers. Each step is of the following form:

$$\begin{aligned} A, B, C, D, E &= (E + f(t, B, C, D) + S_5(A) \\ &\quad + W_t + K_t), A, S^{30}(B), C, D, \end{aligned}$$

where A, B, C, D, E = the five buffer words; t = a step number ($0 \leq t \leq 79$); s^k = circular left shift by k bits; W_t = A 32 bit word derived from the 512-bit message; K_t = An additive constant.

Figure 1 presents basic building block of SHA-1. The compression functions used in SHA-1 are of form shown in Table 1.

2.1 Secure Hash Properties

The Secure Hash Algorithm (SHA) was designed by the National Institute of Standards and Technology (NIST) along with the NSA (National Security Agency) to be used with the Digital Signature Standard. SHA was modelled closely after the MD family of message digest algorithms developed by Rivest. SHA takes a 512 bit input and produces a 160 bit output. SHA also has a 160 bit Initialization Vector (IV) which can be modified but there is a standard setting for this vector which is believed to give good security. SHA was designed to make the process of digitally signing messages more practical. In particular the idea is that instead of signing the entire message, you first apply SHA to the message, get an output of shorter length than the input, and then sign this shorter value which would take less time than signing the original larger message.

We denote by $SHA(IV, x)$ the 160 bit output produced by SHA on a 512 bit user specified input x and the standard IV. SHA is strongly believed to have several fundamental properties which make it an excellent building block for cryptographic protocols and algorithms:

- SHA behaves like a random function: If some significant portion of the input is kept secret, then there is no computationally feasible mechanism for correlating the remaining input bits with the output bits of SHA.

- SHA is collision resistant. That is, it is computationally infeasible to find two distinct 512 bit values, $x_1 \neq x_2$ such that $SHA(IV, x_1) = SHA(IV, x_2)$

- SHA is one way. That is, given $SHA(IV, x)$ it is computationally infeasible to find any x' such that $SHA(IV, x) = SHA(IV, x')$.

- SHA acts as a secure Message Authentication Code: If there is a relatively large (greater than say 128 bits) secret value s, then any adversary who gets to see pairs $(m_1, t_1), ..., (m_2, t_2)$ where each $t_i = SHA(IV, m_i, s)$ will not be able to come up with a pair (m', t') where $t' = SHA(IV, m', s)$ and where m' is different from each of the $m_i's$. Moreover, the adversary will be unsuccesful even if this attack is mounted adaptively; e.g. the adversary can pick a message m_1, be given the corresp onding tag $t_1 = SHA(IV, m_1, s)$, and from this information can pick m_2, see the corresponding t_2 and so on; it will still be impossible for the adversary to come up with a valid (m', t') pair where m' is different from the other m_i.

Table 1: The compression functions used in SHA-1

Step	Primitive Function (g)	g(t,-B,C,D)
$0 \leq t \leq 19$	F_1 (t,B,C,D)	$(B \oplus C) \cup (B' \oplus D)$
$20 \leq t \leq 39$	F_2 (t,B,C,D)	$(B \oplus C \oplus D)$
$40 \leq t \leq 59$	F_3 (t,B,C,D)	$(B \oplus C) \cup (B' \oplus D) \cup (C \oplus D)$
$60 \leq t \leq 79$	F_4 (t,B,C,D)	$(B \oplus C \oplus D)$

These are all well accepted and widely believed properties of SHA. To this day, no one has been able to violate even a single one of these hallowed properties.

3 CPSO Based Algo

There are a number of population based evolutionary computing techniques such as: Evolution strategies [3, 5, 19, 21], genetic algorithms [11, 16]. Particle Swarm Optimization (PSO) is a versatile population-based optimization technique, in many respects similar to evolutionary algorithms. PSO has been shown to perform well for many static problems [17]. As a new stochastic algorithm with quick rate of convergence, CPSO [9] has enjoyed paramount popularity off late.

Optimization with particle swarms has two major ingredients, the particle dynamics and the particle information network. The particle dynamics are derived from swarm simulations in computer graphics, and the information sharing component is inspired by social networks [4, 22]. These ingredients combine to make PSO a robust and efficient optimizer of real-valued objective functions (although PSO has also been successfully applied to combinatorial and discrete problems too). PSO is an accepted computational intelligence technique, sharing some qualities with Evolutionary Computation [10].

In PSO, population members (particles) possess a memory of the best (with respect to an objective function) location that they have visited in the past, pbest, and of its fitness. In addition, particles have access to the best location of any other particle in their own network. These two locations (which will coincide for the best particle in any network) become attractors in the search space of the swarm. Each particle will be repeatedly drawn back to spatial neighborhoods close to these two attractors, which themselves will be updated if the global best and/or particle best is bettered at each particle update. Several network topologies have been tried, with the star or fully connected network remaining a popular choice for unimodal functions. In this network, every particle will share information with every other particle in the swarm so that there is a single gbest global best attractor representing the best location found by the entire swarm.

Particles possess a velocity which influences position updates according to a simple discretion of particle motion.

$$v(t+1) = v(t) + a(t+1) \quad (1)$$
$$x(t+1) = x(t) + v(t+1), \quad (2)$$

where a, v, x and t are acceleration, velocity, position and time (iteration counter) respectively. Equations (1), (2) are similar to particle dynamics in swarm simulations, but PSO particles do not follow a smooth trajectory, instead moving in jumps, in a motion known as a flight [14] (notice that the time increment dt is missing from these rules). The particles experience a linear or spring-like attraction, weighted by a random number, (particle mass is set to unity) towards each attractor.

Explicitly, the acceleration of particle i in Equation (1) is given by:

$$a_i = \chi[\eta\epsilon.(P_g - X_i) + \eta\epsilon.(P_i - X_i)] - (1-\chi)v_i,$$

where ϵ are vectors of random numbers drawn from the uniform distribution $U[0,1]$, $\eta > 2$ is the spring constant and P_i, P_g are particle and global attractors.

4 Our Proposed Algorithm for the Predictor

In our proposed scheme, the canonical particle swarm optimization algorithm (CPSO) is based on the following mathematical relation:

$$v_i^{new} = v_i^{old} + c_1(y_i - x_i)c_2(y_g - x_i)$$
$$x_{i+1} = x_i + v_i^{new},$$

where

$$c_1 = (c_{1f} - c_{1i}) * (iter/M) + c_{1i}$$
$$c_2 = (c_{2f} - c_{2i}) * (iter/M) + c_{2i}$$

where, x_i = The present output of the controller. x_i = The 512 bit transient vector that needs to be bitwise added with the present x_i; y_g denotes global maxima and y_l denotes local maxima.

Figure 2 depicts the building block of our proposed algorithm. Before we embark upon describing our algorithm, we first provide a brief anecdote of our objective function F. A formal mathematical representation of our function is given as:

$$F(h(x_i)) = M,$$

Table 2: Performance comparison between our architecture and conventional one

CPU	Performance in Mb/Sec		Maximum CPU Utilization	
	Normal SHA-1	Our Scheme	Normal SHA-1	Our Scheme
ARM7TDMI	1.92	2.02	82.4%	85.9%
ARM9TDMI	2.08	2.29	78.7%	83.2%
ST22	2.38	2.53	74.3%	78.6%
Pentium III	2.64	2.78	67.8%	71.5%

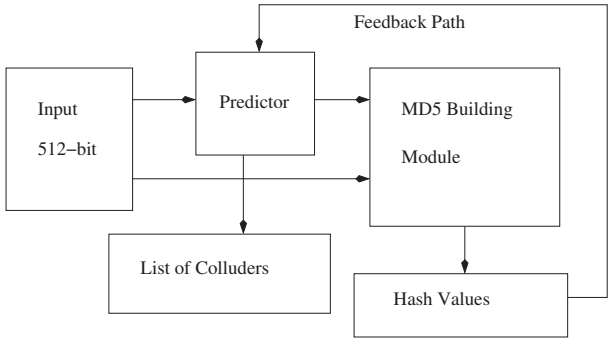

Figure 2: Our proposed predictor based block diagram of SHA-1

where $h(x_i) = 160$ bit hash value generated by MD5 when input is 512-bit output of the i^{th} swarm. $F(h(x_i)) =$ Decimal equivalent of the 128 bit binary value obtained by bit-wise XNOR-ing $h(x_i)$ with that of the hash value of the original message. M is expressed in normalized scale. Also $Y_g =$ Global maxima, or in other words during any iteration it is that 512-bit binary stream which has recorded maximum M value among all the agents in all the iterations up to that. $Y_l =$ Local maxima, or in other words during any iteration it is that 512-bit binary stream which has recorded maximum M value among all the agents in that particular iteration. Now in below we propose our algorithm for the controller of the predictor system.

Algorithm 1 Computing number of colluders Γ

Input: 512 bit Message Stream
Output: Log-List of possible colluders Γ

1: Begin
2: Create a population of 512 swarms.
3: Initiate each of the swarm content with 512-bit long random values.
4: Let F be the optimization function and τ is threshold (0.95 here) value
5: Let β is terminating value (100 here)
6: **for** Each iteration **do**
7: Apply PSO Algorithm, each of the node content is subjected to optimization function F
8: **if** The bit stream for which resultant value $M > \tau$ **then**
9: Note it into log-list
10: **end if**
11: **if** Number of colluders $> \beta$ **then**
12: end for
13: **else**
14: Continue
15: **end if**
16: **end for** Γ
17: End

5 Experimental Results

For simulation and experimental purpose we used 'C' based pseudo-codes for SHA-1 and CPSO algorithms. The stand alone PC used by us was a Pentium4 machine, with a speed of 2.4 GHZ and 248 MB of RAM. First we provide the performance of our novel predictor based scheme, when implemented in different CPU. We compare our architecture with that of a normal SHA-1 architecture.

Table 2 tabulates the performance comparison between our proposed architecture and conventional architecture. Form the Table 2 we can see that employment of our scheme decreases the system speed and increases the CPU utilization to a small extent. But in a trade-off scenario we can overlook them as our scheme can in fact generate the probable colluders list in a surprisingly small number of iteration.

We have tried our scheme with 32 randomly selected 512-bit long message streams. And at the same time we have noted the number of probable colluders it generate after 2^5, 2^{10}, 2^{16} iterations and this is presented in Table 3.

The messages used were shown in Table 4

we can see the predictor generates a substantial number of potential colluders in a very small number of iterations.

6 Conclusion

From the experimental results and other relevant data, it can be vouchsafed that although incorporation of our predictor controller does hamper the system performance and CPU utilization a bit but, with only 2^{16} (compared to 2^{512} in case of brute force analysis) iterations the scheme

Table 3: Number of probable colluders in different iterations

Message Number	After 2^5 iterations	After 2^{10} iterations	After 2^{16} iterations
1	5	61	112
2	3	46	101
3	2	39	90
4	2	58	104
5	4	56	129
6	7	43	145
7	9	52	122
8	3	38	137
9	6	49	143
10	4	70	129
11	3	61	99
12	1	42	106
13	5	56	102
14	4	34	113
15	9	45	129
16	1	56	115
17	1	39	149
18	2	46	138
19	3	53	120
20	1	53	101
21	3	41	110
22	4	58	129
23	6	61	103
24	1	72	150
25	2	59	116
26	3	67	92
27	2	53	108
28	1	36	126
29	3	50	111
30	5	47	142
31	2	29	156
32	4	58	123

Table 4: Used messages as input

	Message in Hex
1	04881D0585845DD1F61E2562242070DB
2	26891E0585845FE3F64F2593042070DB
3	59572D0785845DD1F61F2561232070DE
4	23881E068745AD1F72E2562242070AF
5	07532E3F95845D51F61F2563212DFE68
6	45982F1542315ED1D52E4572242070DB
7	14991A0585845ED1A31F2462143072AC
8	95111F2563745ED1F41D2562741929AA
9	36541E058584543AA64F159304390317
10	F7537E82BD31F2352AD7D2BBEB86D391
11	F4292244432AFF97AB9423A7FC93A039
12	34563287DFEAD23451DDEADECB23546A
13	345167DEAC23AACDEAEACABC456367FA
14	F61E2562C040B340698D5122455AE905
15	4686C62AD9A2E5514ADBCFA9289A7FC5
16	154DFEACBEB768192673EACBBAFCB41
17	518723539FEABCDEB9102837BAEACBDD
18	60423D0795545EDCA42F2561290075AD
19	8B43F6AE6B912219FA579418B6FA5561
20	2381913456ADCEAF27612EACD6187AEC
21	9911234165483ADEEACDEBEAC345127E
22	A9E34563F7523451A123EACABAEA4123
23	214AECC6FA751ADDA655B58C4613F02D
24	A4321738ACCBAEC59572D0785845DD1F
25	BECBADCE2345123881E068745AD1F451
26	8F0BCBE324171FEAC21E1CD34289B7FB
27	24982E068745A45632EE42194207052
28	325BECC6FA751433C21E2C534279B712
29	3219834165483432A3219DEA03240335
30	23881E068745AD1F71E2562242070AF
31	23881E068745AD1F71E2562242070AF
32	10291E2687456534251E55761E20EE32

is able to generate a substantial log list (of about 100-150 components) of potential colluders. Thereby within a very small time it gives the user the power to identify the possible bit-streams which can produce close enough hash values to generate potential hazards such as authentication failure in feature. Although the work is still in progress but it can be expected that in near future predictors which are incorporating stochastic algorithms such as PSO, CPSO, and TVAC-PSO in their controllers will create a whole new avenue in the design of cryptographic hash function.

Acknowledgements

The authors would like to thank the anonymous reviewers for their critical suggestions that greatly improved the quality of this paper.

References

[1] *Advances in Cryptology-Crypto' 99*. Springer-Verlag, 1999.

[2] R. Atkinso. "Security architecture for the internet protocol,". tech. rep., FTF Network working group.

[3] T. Back. "Evolutionary algorithms in theory and practice,". tech. rep., Newyork Oxford University Press.

[4] F. V. D. Bergh and A. P. Englebrecht, "A cooperative approach to particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, pp. 225–239, 2004.

[5] H. G. Beyer, *The Theory of Evolution Strategies*. Springer-Verlag, 2001.

[6] A. Bosselaers, R. Govaerts, and J. Vandewalle, "Fast hashing on the pentium," in *Proceedings of the 16th*

Annual International Cryptology Conference on Advances in Cryptology, pp. 298–312, 1996.

[7] D. Coppersmith, "The data encryption standard (des) and its strength against attack," *IBM Journal of Research and Development*, vol. 38, pp. 243–250, May 1994.

[8] P. A. DesAutels. "Sha1: Secure hash algorithm,". tech. rep., 1997.

[9] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *6th Symposium of Micro Machine and Human Science*, pp. 39–43, Nagoya, Japan, 1995.

[10] A. Engelbrecht, *Computational Intelligence.* John Wiley and Sons, 2002.

[11] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learnin.* Wesley, 1989.

[12] H. Heys and C. Adams, *Selected Areas in Cryptography.* Springer Publications, 2000.

[13] B. Lamacchia, S. Lange, M. Lyons, R. Martin, and K. T. Price, *.Net Framework Security.* Addison-Wesley, 2002.

[14] B. Mandelbrot. "The fractal geometry of nature,". tech. rep., W. H. Freeman and Company, 1983.

[15] A. Menezes. "Handbook on applied cryptography,". tech. rep.

[16] Z. Michalewicz, *Genetic Algorithms: Data Structure, Evolution Programs.* Springer-Verlag, 1994.

[17] K .E. Parsopoulos and M. N. Vrahatis, "Recent approaches to global optimization problems through particle swarm optimization," *Natural Computing*, pp. 235–306, 2002.

[18] B. Preneel, "The state of cryptographic hash functions," in *Lectures on Data Security*, vol. LNCS 1561, pp. 158–182. Springer-Verlag, 1999.

[19] I. Rechenberg, *Evolution Strategy in Compuitational Intelligence: Imitating Life.* IEEE Press, 1995.

[20] M. Robshaw. "Md2, md4, md5, sha and other hash functions,". Tech. Rep. TR-101, RSA Laboratories technical report, 1995.

[21] H. Schwefel, *Numerical optimization of Computer models.* Wiley, 1981.

[22] Y. Shi and A. Khrohling, "Co-evolutionary particle swarm optimization to solve min-max problems," in *Congress on Evolutionary Computation*, pp. 1682–1687, 2002.

[23] J. Touch. "Report on md5 performance,". tech. rep., 1994.

Monjur Alam is presently working as an R & D engineer at Cadence Design Systems (I) Pvt. Ltd. from March 2008. He obtained his B.Tech from the department of Information Technology, Haldia Institute of Technology, Haldia, West Bengal, India in 2005. Subsequently he obtained his M.S. Degree in 2008 from the department of Computer Science and Engg, Indian Institute of Technology Kharagpur. He has published more than 12 technical papers in International Journals and Conferences and has served as Reviewers of several International Conferences and Journals. Monjur was awarded *Instant Recognation Award* several times from Cadence Design Systems. His research interests include cryptography and network security, artificial intelligent, foundation of computer science, etc.

Sonai Ray received his B.Tech degree in Computer Science and Engineering from Jadavpur University, Jadavpur, West Bengal, India in 2005. He has been working in several EDA industries since then. His research interests include EDA tools development, cryptography and network security, artificial intelligent, etc. He has authored about 8 technical papers in International Conferences.