HF-hash: Hash Functions Using Restricted HFE Challenge-1

Dhananjoy Dey[1,2], Prasanna Raghaw Mishra[1] and Indranath Sengupta[2,*]

[1] *SAG, Metcalfe House, Delhi-110 054, India*
[2] *Department of Mathematics, Jadavpur University, Kolkata, WB 700 032, India*
{ddey06, prasanna.r.mishra, sengupta.indranath} @gmail.com

Abstract

Vulnerability of dedicated hash functions to various attacks has made the task of designing hash function much more challenging. This provides us a strong motivation to design a new cryptographic hash function viz. HF-hash. This is a hash function, whose compression function is designed by using first 32 polynomials of HFE Challenge-1 [8] with 64 variables by forcing remaining 16 variables as zero. HF-hash gives 256 bits message digest and is as efficient as SHA-256. It is secure against the differential attack proposed by Chabaud and Joux in [6] as well as by Wang et. al. in [25] applied to SHA-0 and SHA-1. We have also compared the efficiency of our HF-hash with SHA-256.

Keywords: *Collision search attack, dedicated hash functions, differential attack, HFE challenge*

1. Introduction

The majority of dedicated hash functions published are more or less designed using ideas inspired by hash functions MD4 [20] and MD5 [21]. Not only the hash functions HAVAL [28], RIPEMD [3], RIPEMD-160 [18] but also SHA-0 [15], SHA-1 [16] and SHA-2 family [17] are designed using the similar ideas. The hash functions HAS-160 [23] and HAS-V [19] both exhibit strong resemblance with SHA-1.

While comparing compression functions of the aforementioned hash functions it is easy to observe that all of them have the three fundamental parts viz. *the message expansion algorithm* which is required for creating more disturbance pattern for the input to the compression function, *the iteration of the step transformation* which is required for taking arbitrary length of input and *the state feed-forward operation* which is required for updating the chaining variables or the internal hash value.

The most commonly used dedicated hash functions are MD5 and SHA-1. The first member of the MD family, viz. MD4 was published in 1990. After one year, an attack on the last two out of three rounds has been presented in [1]. After that Rivest designed the improved version of MD4, called MD5. Later, Vaudenay showed that the first two rounds of MD4 are not collision-resistant and it is possible to get near-collisions for the full MD4 [24].

In 1993, Boer and Bosselaers [2] showed that it is possible to find pseudo-collisions for the compression function of MD5, i.e. they showed a way of finding two different values of the initial value *IV* for the same message *M* such that

$$MD5-compress(IV,M)=MD5-compress(IV',M).$$

[*] This work was done while the author was visiting the Ramakrishna Mission Vivekananda University, Belur Math, Howrah, WB 711 202, INDIA on Lien from the Jadavpur University.

This was the first attack on MD5. This did not threaten the usual applications of MD5, since in normal situations one cannot control inputs of chaining variables.

A major step forward in the analysis of MD-based designs was made by H. Dobbertin who developed a general method of attacking designs similar to MD4 in 1996. His method aims at finding collisions and is based on describing the function as a system of complicated, non-linear equations that represent the function. With this method he successfully attacked MD4 showing that one can find collisions using computational effort of around 2^{20} hash evaluations [10]. He also showed collisions for the compression function of MD5 with a chosen *IV* [11].

The other family of dedicated hash function is SHA family. The first version of the Secure Hash Algorithm (SHA) i.e. SHA-0 was presented by NIST in 1993. Two years later, this function was slightly modified and an updated version of the standard was issued in 1995. Indeed, in 1998 Chabaud and Joux presented a differential attack on the initially proposed function, SHA-0, that can be used to find collisions with complexity of 2^{61} hash evaluations. Since SHA-0 and SHA-1 are different by a small change in the message expansion algorithms, it is quite natural question to ask whether it is possible to extend the original attack of Chabaud and Joux to the improved design of SHA-1. Due to the same round structure, the same technique used to attack SHA-0 could be applied to launch an attack on SHA-1 provided there exists a good enough differential pattern. Novel ideas of Wang et al. contributed a lot in opening new avenues of analysis of SHA-1. It seems the ability to influence the value of the new word of the state in each step combined with rather weak message expansion algorithms is the fundamental weakness of designs of that family that can be exploited that way or another.

In August 2002, NIST announced a new standard FIPS 180-2 that introduced three new cryptographic hash functions viz. SHA-256, SHA-384 and SHA-512. In 2004 the specification was updated with one more hash, SHA- 224. All these algorithms are very closely related. In fact SHA-224 is just SHA-256 with truncated hash and SHA-384 is a truncated version of SHA-512. These are called the SHA-2 family of hashes. The design of SHA-512 is very similar to SHA-256, but it uses 64-bit words and some parameters are different to accommodate for this change. Clearly, the fundamental design of this family is SHA-256 and all the other algorithms are variations of that one, so the question of the security of SHA-256 is an extremely interesting one.

We have designed a new hash function HF-hash using the restricted version of HFE Challenge-1 as the compression function, which gives 256 bits message digest. We have used the first 32 equations of HFE Challenge-1 with first 64 variables by setting remaining 16 variables to zero. Although the first proposal of designing hash function using quadratic or higher degree multivariate polynomials over a finite field as the compression function was given by Billet et. al. [4] as well as by Ding and Yang [12] in 2007, they did not present how to design a secure hash function. In these papers they have used multivariate polynomials for both cases viz. message expansion as well as message compression.

The compression function of HF-hash depends on the following well-known facts:

- Computing the values of a random set of *m* multivariate polynomials in *n* variables over a finite field $\boldsymbol{F}$ viz., $(p_1(x_1,\cdots,x_n),\cdots, p_m(x_1,\cdots,x_n))$, for any fixed $(x_1, \cdots, x_n)$ is easy.
- Finding a solution of this set of polynomial equations is an NP-hard problem[†] [13].

[†] It is true even if we restrict the total degree of these polynomials to at least 2.

The expansion procedure of *HF-hash* for a message block is very much similar to the message expansion of SHA family but it differs in padding and parsing procedure from that of SHA family.

In this paper we present a complete description of HF-hash, and its analysis in the subsequent sections.

2. HF-hash

HF-hash function can take arbitrary length $(<2^{64})$ of input and gives 256 bits output. We have designed an iterative hash function, which uses restricted HFE Challenge-1 [8] as compression function. The hash value of a message *M* of length *l* bits can be computed in the following manner:

Padding: First we append 1 to the end of the message *M*. Let *k* be the number of zeros added for padding. The 64-bit representation of *l* is appended to the end of k zeros. The padded message *M* is shown in the following figure. Now *k* will be the smallest positive integer satisfying the following condition:

$$l+1+k+64\equiv 0 \bmod 448$$
$$i.e.,\quad k+l\equiv 383 \bmod 448$$

Figure 1. Padded Message *M*

Parsing: Let l' be the length of the padded message. Divide the padded message into $n(=l'/448)$ 448-bit block i.e. fourteen 32-bit words. Let $M^{(i)}$ denote the i^{th} block of the padded message, where $1\le i\le n$ and each word of i^{th} block is denoted by $M_j^{(i)}\ for 1\le j\le 14.$

Initial Value: Take the first 256 bits initial value i.e., eight 32-bit words from the expansion of the fractional part of π and hexadecimal representation of these eight words are given below:

$$h_0^{(0)}=243F6A88, h_1^{(0)}=85A308D3, h_2^{(0)}=13198A2E, h_3^{(0)}=03707344,$$
$$h_4^{(0)}=A4093822, h_5^{(0)}=299F31D0, h_6^{(0)}=082EFA98, h_7^{(0)}=EC4E6C89.$$

Hash Computation: For each 448-bit block $M^{(1)}, M^{(2)}, \cdots, M^{(n)}$, the following four steps are executed for all the values of *i* from 1 to *n*.

1. **Initialization**

$$H_j = h_j^{(i-1)}\ \ for\ 0\le j\le 7.$$

2. Expansion

i. $W_0 \leftarrow H_0$

ii. $W_j \leftarrow M_j^{(i)},\ for\ 1 \le j \le 14$

iii. $W_{15} \leftarrow H_7$

iv. $W_j \leftarrow rotl_3(W_{j-16} \oplus W_{j-14} \oplus W_{j-8} \oplus W_{j-1}),\ for\ 16 \le j \le 63,$ where $rotl_k$ denotes the left rotation by *k*.

This is the expansion of the message blocks without padding. In the last block we apply padding rule. If $(l+1) > 384$ bits, then we have two extra blocks in the padded message. Otherwise we have one extra block in the padded message. In both the cases, we apply the following expansion rule for the last block so that the length of the message appears in the end of the padded message.

i. $W_0 \leftarrow H_0$

ii. $W_1 \leftarrow H_7$

iii. $W_j \leftarrow M_j^{(i)},\ for\ 2 \le j \le 15$

iv. $W_j \leftarrow rotl_3(W_{j-16} \oplus W_{j-14} \oplus W_{j-8} \oplus W_{j-1}),\ for\ 16 \le j \le 63$

3. Iteration for *j* = 0 to 63

i. $T_1 \leftarrow H_1 + H_2 + p(H_3 \| H_0) + K_j$ ‡

ii. $T_2 \leftarrow H_4 + H_5 + p(H_7 \| H_6) + W_j$

iii. $H_7 \leftarrow H_6$

iv. $H_6 \leftarrow H_5$

v. $H_5 \leftarrow H_4$

vi. $H_4 \leftarrow rotl_5(H_3 + T_2)$

vii. $H_3 \leftarrow H_2$

viii. $H_2 \leftarrow H_1$

ix. $H_1 \leftarrow H_0$

x. $H_0 \leftarrow T_1 + T_2$, where T_1 and T_2 are two temporary variables and $p : Z_{2^{64}} \rightarrow Z_{2^{32}}$ be a function defined by $p(x) = 2^{31}.p_1(x_1, \cdots, x_{64}) + 2^{30}.p_2(x_1, \cdots, x_{64}) + \cdots + 1.p_{32}(x_1, \cdots, x_{64})$. Since any element $x \in Z_{2^{64}}$ can be represented by $x_1 x_2 \cdots x_{64}$, where $x_1 x_2 \cdots x_{64}$ denotes the bits of *x* in decreasing order of their significance. The polynomial $p_i(x_1, \cdots, x_{64})$ denotes the i^{th} polynomial of HFE challenge-1 with 64 variables by setting the remaining 16 variables to zero for $1 \le i \le 32$ and these polynomials are found in *http://arxiv.org/PS cache/arxiv/pdf/0909/0909.1392v2.pdf*

‡ The operation || denotes the concatenation and + denotes the addition *mod* 2^{32}.

The 64 constants K_j are taken from the fractional part of e and are given below:

K_0 = AC211BEC	K_1 = 5FEFE110	K_2 = 112276F8	K_3 = 8AE122A4
K_4 = 18B3488B	K_5 = 00921A36	K_6 = 40C045F8	K_7 = C8C0A3DA
K_8 = C4ABF676	K_9 = 6A68C750	K_{10} = A37AFE0F	K_{11} = 732806F3
K_{12} = 25722CB7	K_{13} = 3FF43825	K_{14} = ACDF96D7	K_{15} = 9B53BCD3
K_{16} = E34950DE	K_{17} = D9780CCB	K_{18} = 8B5F9BB7	K_{19} = 3D1182ED
K_{20} = 1921B44A	K_{21} = 7003F30D	K_{22} = 42657E31	K_{23} = 231E7B55
K_{24} = 91E3A28E	K_{25} = 95CD4AB0	K_{26} = 0A0AC2E3	K_{27} = FCDEBE5E
K_{28} = FCF1E321	K_{29} = 1D136560	K_{30} = 2974BF63	K_{31} = 70963992
K_{32} = 4F5B5107	K_{33} = 0072C0C1	K_{34} = C99F3C1D	K_{35} = C56598D9
K_{36} = 77A1D027	K_{37} = 36675FB6	K_{38} = A40C34E8	K_{39} = 46764EAD
K_{40} = F8823861	K_{41} = 19F66E64	K_{42} = 87E10299	K_{43} = 4311C8C2
K_{44} = 07C102B9	K_{45} = 9F4EC8CE	K_{46} = 29D81EBA	K_{47} = 992744F9
K_{48} = 4CDA6790	K_{49} = 13DA5357	K_{50} = BA6D7772	K_{51} = 80673F08
K_{52} = B049EE4C	K_{53} = 839F8647	K_{54} = 736F658B	K_{55} = EBE90F9B
K_{56} = FA6DC4D1	K_{57} = E951630E	K_{58} = AFC453E4	K_{59} = 159B7483
K_{60} = 45EABF9D	K_{61} = 4292A60E	K_{62} = 17AA0ABD	K_{63} = 94E81C30

4. Intermediate Hash Value

The i^{th} intermediate hash value

$$h^{(i)} = h_0^{(i)} \| h_1^{(i)} \| h_2^{(i)} \| h_3^{(i)} \| h_4^{(i)} \| h_5^{(i)} \| h_6^{(i)} \| h_7^{(i)},$$

where $h_j^{(i)} = H_j \ \ for\ 0 \le j \le 7.$ This $h^{(i)}$ will be the initial value for the message block $M^{(i+1)}$.

The final hash value of the message *M* will be

$$h_0^{(n)} \| h_1^{(n)} \| h_2^{(n)} \| h_3^{(n)} \| h_4^{(n)} \| h_5^{(n)} \| h_6^{(n)} \| h_7^{(n)},$$

where $h_i^{(n)} = H_i \ \ for\ 0 \le i \le 7.$

Process of Implementation: In order to compute *HF-hash*(*M*), first the padding rule is applied and then the padded message is divided into 448-bit blocks. Now each 448-bit block is divided into fourteen 32-bit words and each 32-bit word is read in *little endian* format. For example, suppose we have to read an ASCII file with data *'abcd'*, it will be read as *0x64636261*.

Test Value of HF-hash: Test values of the three inputs are given below:

HF-hash(*a*)	=	*04EAF5F6*	*B215D974*	*B827FCC2*	*5ECA45C3*
		031524E8	*472617D1*	*C14D9C85*	*6ACD1DC3*
HF-hash(*ab*)	=	*F2DD83C8*	*34E96291*	*E39040B9*	*BCD3E624*
		BA01846E	*0D5E5083*	*492DC4BF*	*C0720235*
HF-hash(*abc*)	=	*E9582019*	*216033AA*	*346E8D46*	*11D131A7*
		D0635A5E	*92D5B13D*	*2DC481B8*	*836774B6*

3. Analysis of HF-hash

In this section we will present the complete analysis of *HF-hash*, which includes properties, efficiency, as well as the security analysis of this function.

3.1. Properties of HF-hash

This subsection describes the properties of *HF-hash* required for cryptographic applications.

1. **Easy to compute:** For any given value x it is easy to compute *HF-hash*(x) and the efficiency of this hash function is given in section 3.2.
2. **One-wayness:** Suppose one knows the *HF-hash*(x) for an input x. Now to find the value of x, (s)he has to solve the system of polynomial equations consisting of 32 polynomials with 64 variables for each round operation. Since this system of equations is *underdefined* therefore *XL* method [7] or any variant of *XL* [27] cannot be applied to solve this system.

   Now if one wants to solve this system of equations using the *Algorithm A*[§] given by Courtois et. al. in [5], then at least 2^{25} operations are required to solve for one round of *HF-hash*. Since *HF-hash* has 64 rounds one has to compute $2^{25\times 64}$ operations to get back the value of x for given *HF-hash*(x). This is far beyond the today's computation power. Thus, for any given *HF-hash*(x) it is difficult to find the input x.
3. **Randomness:** We have taken an input file M consisting of 448 bits and computed *HF-hash*(M). Then 448 flies M_i are generated by changing the i^{th} bit of M for $1 \le i \le 448$. Then computed *HF-hash*(M_i) of all the 448 files and calculated the Hamming distance d_i between *HF-hash*(M) and *HF-hash*(M_i) for $1 \le i \le 448$ as well as the distances between corresponding eight 32-bit words of the hash values. The following table shows the *maximum*, the *minimum*, the *mode* and the *mean* values of the above distances.

Table 1

Changes	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	*HF-hash*
Max	25	24	24	26	25	23	23	24	149
Min	6	7	7	8	7	8	9	8	103
Mode	14	17	17	16	16	17	16	15	132
Mean	16	16	16	16	16	16	16	16	128

For ideal case d_i should be 128 for $1 \le i \le 448$. But we have found that d_i's were lying between 103 and 149 for the above files. The following bar chart and the table show the distribution of above 448 files with respect to their distances.

[§] Algorithm A is the best algorithm for solving our system of equations among Algorithms A, B & C.

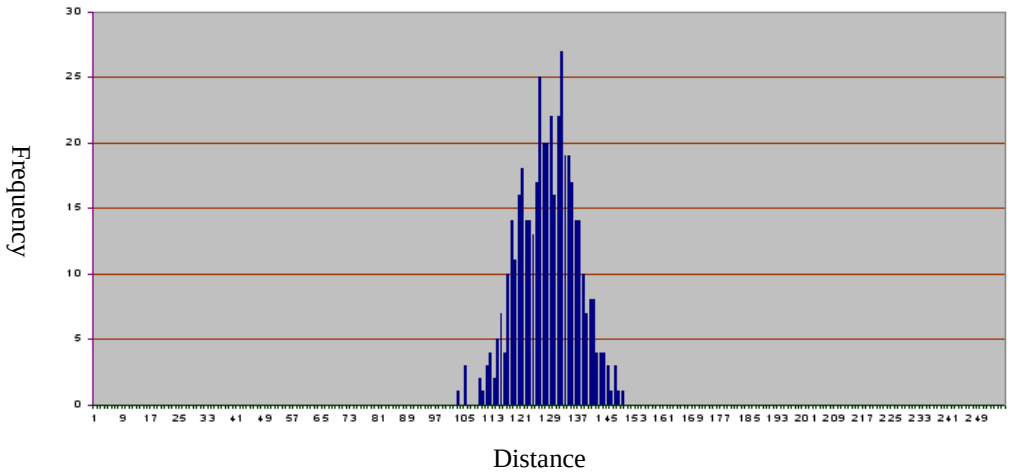

Figure 2. Frequency Distribution

Table 2

Range of Distance	No. of Files	Percentage
128 ± 5	215	47.99
128 ± 10	362	80.80
128 ± 15	421	93.97
128 ± 20	443	98.88

The above analyses show that HF-hash exhibits a reasonably good avalanche effect. Thus it can be used for cryptographic applications.

3.2. Efficiency of HF-hash

The following table gives a comparative study in the efficiency of *HF-hash* with SHA-256 in HP Pentium - D with 3 GHz processor and 512 MB RAM.

Table 3

File Size (in MB)	SHA-256 (in Sec.)	***HF-hash* (in Sec)**
1.4	18.64	**20.02**
4.84	60.08	**67.72**
7.48	103.59	**109.73**
12.94	169.19	**181.01**
24.3	313.53	**345.53**

Although, SHA-256 is little bit faster than *HF-hash* but *HF-hash* is more secure than SHA-256 in case of either collision search or differential attack. Since the design

principle of SHA-256 is almost similar to that of SHA-1, therefore all the attacks applied to SHA-1 can also be extended to SHA-256.

3.3. Security Analysis

In this paper we have applied a new method for expanding a 512-bit message block into 2048-bit block. For this purpose we have to change the padding rule and the procedure of parsing a padded message. In case of MD-5, SHA-1 and SHA-256, the padded message is divided into 512-bit blocks whereas in case of *HF-hash*, the padded message is divided into 448-bit blocks. Then two 32-bit words are added to construct a 512-bit block as the input for each iteration, where these two words depend on the previous internal hash updates or chaining variables. So, in each iteration, the 512-bit blocks are not independent from the previous message blocks as in the case of MD-5, SHA-1 or SHA-256. Message expansion algorithm of *HF-hash* is dependent on the first and last word of the previous hash. Now if small change is occurred in the inputs, the intermediate hash values will be different. Thus we will get the differences in first and last words of intermediate hash values. These differences along with the rotation in the message expansion formula make impossible to find corrective pattern described in [6]. Thus, differential attack by Chabaud and Joux is not applicable to our hash function because one does not have any control over two 32-bit words coming from the previous internal hash updates.

Moreover, a 1-bit difference in any one of fourteen initial 32-bit words propagates itself to at least 165 bits of the expanded message since we have taken the 64 round operations. Less than 75 bit difference in expanded message and input message is obtained by changing 1-bit input when 32 or 48 round operations are performed. That is why we have taken 64 round operations for *HF-hash* function. This makes it impossible to find corrective patterns used by Chabaud and Joux in [6], due to the reason that differences propagate to other positions.

The idea of Wang et. al. for finding collision in SHA-0 [26] and SHA-1 [25] is to find out the *disturbance vectors* with low Hamming weight first and then to construct a *differential path*. To construct a valid differential path, it is important to control the difference propagation in each chaining variable. After identifying the wanted and unwanted differences one can apply the Boolean functions (mainly *IF*) and the carry effect to cancel out these differences. In particular, when an input difference is 1, the output difference can be 1, -1 or 0. Hence, the function can preserve, flip or absorb an input difference. This gives a good flexibility to construct a differential path. The key of these attacks was the Boolean functions used in compression function, which in combination with carry effect facilitate the differential attack.

We have replaced the Boolean functions with restricted hidden field polynomials. Now if we change 1 bit in the inputs of HF-hash, the outputs will be the same after one round of operation of the compression function. Because, this input difference will not effect since in our case $W_0 = H_0$. But this input difference will appear in W_1. Thus, the output differences will be found after two rounds of computing compression function. We have computed the difference propagation of chaining variables for several files having 1 bit input difference and the result is given in the following table.

Table 4

Round	Minimum	Maximum
2	35	53
3	63	134
4	88	144
5	104	145

This shows that it is impossible to control the difference propagation of chaining variable after round two. Therefore, these attacks are not applicable to our hash function.

Although the cross dependence equation described by Sanadhya and Sarkar in [22] can be formed in case of *HF-hash*, the procedure of message expansion as well as the compression function of *HF-hash* being different from SHA-2 family, this procedure for finding collision cannot be applied in our hash function. Thus, this hash function is also collision resistance against the method described by Sanadhya and Sarkar.

Thus the compression function of *HF-hash* is collision-resistant against existing attacks. Since *IV* of *HF-hash* is fixed and the padding procedure of *HF-hash* includes the length of the message, therefore by Merkle-Damgard theorem [9] [14] we can say that *HF-hash* is collision resistant against existing attacks.

4. Conclusions

In this paper a dedicated hash function *HF-hash* has been presented. The differential attack applied by Chabaud and Joux in SHA-0, collision search for SHA-1 by Wang et. al. as well as collision search method applied by Sarkar et. al. for SHA-2 family are not applicable to this hash function. The main differences of HF-hash with MD family and SHA family lie in the procedure of message expansion and the compression function. A system of multivariate polynomials taken from HFE challenge-1 (restricted form) is used for designing the compression function of this hash function. Analysis of this hash functions viz. randomness as well as security proof are also described here.

The system of equations in HFE challenge-1 is neither regular system nor the minimal set of polynomials. Presently we are looking at the behavior of *HF-hash* when the minimal system or the Groebner basis of the ideal generated by the above system or randomly selected 32 polynomials with 64 variables is taken.

References

[1] B. Boer and A. Bosselaers, "*An Attack on the Last Two Rounds of MD4*", in Advances in Cryptology – CRYPTO '91, LNCS 0576, pages 194 - 203, Springer-Verlag, 1991.

[2] B. Boer and A. Bosselaers, "*Collisions for the Compression Function of MD5*", in Advances in Cryptology EUROCRYPT '93, LNCS 0765, pages 293 - 304, Springer-Verlag, 1994.

[3] A. Bosselaers and B. Preneel, editors, "*Integrity Primitives for Secure Information Systems*", Final Report of RACE Integrity Primitives Evaluation, LNCS 1007, Springer-Verlag, 1995.

[4] O. Billet, M. Robshaw and T. Peyrin, "*On Building Hash Functions from Multivariate Quadratic Equations*", ACISP, LNCS 4586, pages 82 - 95, Springer, 2007.

[5] N. Courtois, L. Goubin, W. Meier and J. Tacier, "*Solving Underdefined Systems of Multivariate Quadratic Equation*", PKC '02, LNCS 2274, pages 211 - 227, Springer-Verlag, 2002.

[6] F. Chabaud and A. Joux, "*Differential Collisions in SHA-0*", in Advances in Cryptology – CRYPTO '98, LNCS 1462, pages 56 - 71, Springer-Verlag, 1998.

[7] N. Courtois, A. Klimov, J. Patarin and A. Shamir, "*Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations*", EUROCRYPT '2000, LNCS 1807, pages 392 - 407, Springer-Verlag, 2000.

[8] N. Courtois, "*Hidden Field Equation Challenge – 1*", 1998. Available online at www.univ-tln.fr/~courtois/hfe.html

[9] I. B. Damgard, "*A Design Principle for Hash Functions*", in Advances in Cryptology CRYPTO '89, pages 416 - 427, Springer-Verlag, 1990.

[10] H. Dobbertin, "*Cryptanalysis of MD4*", in Fast Software Encryption FSE '96, LNCS 1039, pages 53 - 69, Springer-Verlag, 1996.

[11] H. Dobbertin, "*Cryptanalysis of MD5*", Presented at the rump session of EUROCRYPT '96, pages 12 - 16, 1996.

[12] J. Ding and B. Yang, "*Multivariate Polynomials for Hashing*", in Cryptology ePrint Archive, Report 2007/137, 2007. Available online at http://eprint.iacr.org/2007/137.pdf

[13] M. Garey and D. Johnson, "*Computers and Intractability - A Guide to the Theory of NP-completeness*", Freeman and Co., 1979.

[14] R. C. Merkle, "*One-way Hash Functions and DES*", in Advances in Cryptology CRYPTO '89 pages 428 - 446, Springer-Verlag, 1990.

[15] National Institute of Technology, "*Secure Hash Standard*", FIPS Publication-180, 1993.

[16] National Institute of Technology, "*Secure Hash Standard*", FIPS Publication-180-1, 1995.

[17] National Institute of Technology, "*Secure Hash Standard*", FIPS Publication-180-2, 2002.

[18] B. Preneel, A. Bosselaers and H. Dobbertin, "*RIPEMD-160: A Strengthened Version of RIPEMD*", in Fast Software Encryption FSE '96, LNCS 1039, pages 71 - 82, Springer-Verlag, 1997.

[19] N. K. Park, J. H. Hwang and P. J. Lee, "*HAS-V: A New Hash Function with Variable Output Length*", in Selected Areas in Cryptography SAC '00, LNCS 2012, pages 202 - 216, Springer-Verlag, 2000.

[20] R. L. Rivest, "*The MD4 Message Digest Algorithm*", in Advances in Cryptology – CRYPTO '90, LNCS 0537, pages 303 - 311, Springer-Verlag, 1991.

[21] R. L. Rivest, "*The MD5 Message Digest Algorithm*", Request for Comments (RFC) 1321, Internet Engineering Task Force, April 1992.

[22] S. Sanadhya and P. Sarkar, "*Attacking Step Reduced SHA-2 Family in a Unified Framework*", in Cryptology ePrint Archive, Report 2008/271, 2008. Available online at http://eprint.iacr.org/2008/271.pdf

[23] Telecommunications Technology Association of Korea, "*Hash Function Standard Part 2: Hash Function Algorithm Standard* (*HAS-160*)", TTA Standard TTAS.KO-12.0011, 1998. Available online at http://www.tta.or.kr/English/new/standardization/engttastddesc.jsp?stdno=TTAS.KO-12.0011

[24] S. Vaudenay, "*On the Need for Multi-permutations: Cryptanalysis of MD4 and SAFER*", in Fast Software Encryption FSE '94, LNCS 1008, pages 286 297, Springer-Verlag, 1995.

[25] X. Wang, Y. Yin and H. Yu, "*Finding Collisions in the Full SHA-1*", in Advances in Cryptology CRYPTO '05, LNCS 3621, pages 17 - 36, Springer, 2005.

[26] X. Wang, H. Yu and Y. Yin, "*Efficient Collision Search Attacks on SHA-0*", in Advances in Cryptology CRYPTO '05, LNCS 3621, pages 1 - 16, Springer, 2005.

[27] P. Yang and J. Chen, "All in the XL Family: Theory and Practice", Available online at http://precision.moscito.org/by-pub/recent/xxl.pdf

[28] Y. Zheng, J. Pieprzyk, and J. Seberry, "*HAVAL - a One-way Hashing Algorithm with Variable Length of Output*", in Advances in Cryptology AUSCRYPT '92, LNCS 0718, pages 83 - 104, Springer-Verlag, 1993.

Authors

Dhananjoy Dey received his Master degree in Mathematics from Jadavpur University in 1998. After that he joined DRDO in 2000 at SAG, Delhi, INDIA. Currently he is pursuing his PhD from Jadavpur University. His current research interests include the analysis of multivariate public key cryptography and the design and analysis of cryptographic hash functions and block ciphers.

Dr. Prasanna Raghaw Mishra did his Master degree in Mathematics from VBS Purvanchal University in 1997. He received his PhD degree in Number Theory from Banaras Hindu University. He joined DRDO in March 2003 and was posted to SAG in August 2003. His current research interests include the design and analysis of cryptographic hash functions and the crypto-algorithms.

Dr. Indranath Sengupta received his Ph.D. degree from the Indian Institute of Science, Bangalore, INDIA in the year 2001. He is now a Reader in the department of Mathematics, Jadavpur University, where he has been teaching since 2001. His current research interest is Commutative Algebra, Algebraic Geometry and its applications to Cryptology.