A New Design of Cryptographic Hash Function: Gear

Abdulaziz M Alkandari[1], Khalil Ibrahim Alkandari[2], Imad Fakhri Alshaikhli[3], Mohammad A. AlAhmad[4]

[1]Public Authority for Applied Education and Training
College of Technology, Science Department, Kuwait City, Kuwait
abdl_alkandari@hotmail.com
[2]Public Authority for Applied Education and Training
The Higher Institute of Telecommunication &Navigation
Computer Department, Kuwait City, Kuwait
ki.alkandari@paaet.edu.kw
[3]Department of Computer Science, International Islamic University of Malaysia, 53100 Jalan Gombak
Kuala Lumpur, Malaysia
imadf@iium.edu.my
[4]Public Authority for Applied Education and Training College of Basic Education,
Computer Science Department, P.O. Box 34567 Adailiyah, 73205, Kuwait City, Kuwait
malahmads@yahoo.com

Abstract— A hash function is any function that can be used to map data of arbitrary size to data of fixed size. A hash function usually has two main components: a permutation function or compression function and mode of operation. We will propose a new concrete novel design of a permutation based hash functions called Gear in this paper. It is a hash function based on block cipher in Davies-Meyer mode. It uses the patched version of Merkle-Damgård, i.e. the wide pipe construction as its mode of operation. Thus, the intermediate chaining value has at least twice larger length than the output hash. And the permutations functions used in Gear are inspired from the SHA-3 finalist Grøestl hash function which is originally inspired from Rijndael design (AES). There is a very strong confusion and diffusion in Gear as a result.

Keywords— WP - permutation –block cipher – AES

I. Introduction

To understand how to use and retail functions to verify the integrity and source of information, you must first examine the characteristics and the origin of the basic function retail. The standard hash function serves as a basis for the discussion of

Cryptographic Hash Functions. Cryptographic hash functions have indeed proved to be the workhorses for modern cryptographic hash functions. Another name given to cryptographic hash functions is “Swiss knife army” because it can serve many different purposes such as digital signatures, conventional message authentication to secure passwords storage or forensics data identification. Cryptographic hash functions take an unfixed size of input and produce a fixed size of an output.

A hash function usually built from two main components: (1) a basic primitive compression function C and (2) an iterative mode of operation H, where the symbol HC denotes the hash function HC based on the compression function C. Most hash functions in use today are so-called iterated hash functions, i.e. Merkle-Damgård (MD), based on iterating a compression function. Examples of iterated hash functions are MD4, MD5, SHA and RIPEMD-160. For a cryptographic hash function HC, if the compression function C is resistant to the following attacks, then the hash function considered secure:

- Preimage: given y = H(x), find x’such that H(x’) = y,
- 2nd preimage: given an x and y=H(x) find x’≠ x such that H(x’) = y,
- Collision: find x and x’such that x’≠ x and H(x) =H(x’).

Recently, several collisions were announced which decreased the security of some of the existing hash functions. Particularly, collisions were announced in SHA-0, MD4, MD5, HAVAL-128, and RIPEMD. French researcher Antoine Joux et al. [17] presented the collision in SHA-0, and a group of collisions against MD4, MD5, HAVAL-128, and RIPEMD were found by the Chinese researcher Xiaoyun Wang with co-authors Dengguo Feng, Lai, and Hongbo Yu [30]. After that, in February 2005, the same Xiaoyun Wang, Lisa Yiqun Yin, and Hongbo Yu found collisions in SHA-1 using 269 hash computations [30]. Several strategies were developed to thwart these

attacks. Stefan lucks et al. [23] introduced the Wide Pipe (WP) hash construction as an intermediate version of Merkle-Damgård to improve the structural weaknesses of Merkle-Damgård design. The process is similar to Merkle-Damgård algorithm steps except of having a larger internal state size, which means the final hash digest is smaller than the internal state size of bit length. For example, the final compression function compresses the internal state length (for ex, 2n-bits) to output a hash digest of n-bit. This simply can be achieved by discarding the last half of 2n-bit output. WP is used in this paper to construct Gearhash function. It is used as an operation of mode for Gear. Mridul Nandi and Souradyauti Paul et al. [31] proposed the fast wide pipe (FWP) construction to overcome these attacks. It is twice faster than the wide pipe construction. HAsh Iterated FrAmework (HAIFA) is also a patched version Merkle-Damgård construction [32]. HAIFA design solves many of the internal collision problems associated with the classic MD construction design by adding a fixed (optional) salt of s-bits along with a (mandatory) counter Ci of t-bits to every message block in the iteration i of the hash function. Wide-pipe and HAIFA are very similar designs. Where, sponge construction is an iterative construction designed by Guido Bertoni, Joan Daemen, MichealPeeter and Gilles Van Assche to replace Merkle-Damgård construction [2]. It is a construction that maps a variable length input to a variable length output. Keccak (SHA-3 winner) hash function uses sponge construction. In the next section, we demonstrate our new proposal Gear hash function in more details.

II. OUR PROPOSAL

We propose a new hash function called Gear that supports 256-512 bits digests. The basic building block of our hash is a block cipher. By applying standard design approaches next we create a compression function (based on the cipher), and finally a hash function. We use the following design techniques:

- The block cipher applies the wide trail strategy.
- A compression function based on the block cipher in Davies-Meyer mode.
- A hash built upon an iterative compression function with the Merkle-Damgård construction.
- A wide pipe construction, i.e. the intermediate chaining value has at least twice larger length than the output hash.

III. DESIGN GOALS

In the last several years, the notion of security has expanded to include not only the basic requirements on collisions and second preimage resistance, but also a wide variety of distinguishers. So, the main design goal of any modern hash function is the security of the construction. In fact, non-formally a hash function is supposed to behave as a random oracle. Although in this model, trivial distinguishers do exist for every hash function, the designers aim to construct hash function that will be resistant against all possible non-trivial distinguishers, i.e. the hash does not exhibit any structural distinguishers, and, in a line of notation from the Sponge design [2], it is a hermetic design.

We aim to achieve this high security requirement with our proposal as well. More precisely, we would like to achieve the standard security margin against the following attacks and structural distinguishers:

- No collisions can be found in n-bit Gear with significantly less than 2n hash function invocations
- No (second) preimage can be found in n-bit Gear with significantly less then 2n invocations
- No non-trivial structural distinguishers can be found for Gear with a complexity significantly lower than the complexity required to find (or confirm) such property in a secure hash function (such as SHA-2, SHA-3, etc.)Here, we would like to point out that the deviation “significantly lower” from “lower” is introduced to annulate the analysis based on the recently discovered bicliques[7] – the latest results suggest that such analytical results are most likely applicable to all cryptographic primitives, thus one cannot expect the achieve the ideal security level. On the other hand, the complexity of the attacks not based on granulation of the compression function (i.e. all other analysis except bicliques), should always exceed our claimed security bound.

IV. DESCRIPTION OF GEAR

Our proposal Gearis a wide pipe hash function with an internal state of 1024 bits. It supports digests of 1 to 512 bits. For security reasons, we suggest a minimal output of 256 bits – further we describe the two main versions Gear- 256 and Gear-512, with an output length of 256 and 512

bits, respectively. We emphasize that these two versions, as well as all the possible versions with a hash output between 256 and 512 bits, are based on the same primitive, and differ only in the number of bits that are truncated at the output of the primitive. Our hash function is based on a cipher C-Gear used in the Davies-Meyer mode to build a compression function. We use Merkle-Damgård to construct the hash upon this compression function. Further we describe in details the cipher and give a brief recall of the mode.

A. The Cipher C-Gear

The block cipher C-Gear(P, K) is an SP network with 16 rounds and designed according to the wide trail strategy. It has a state of 1024 bits and supports 1024-bit keys. The state as well as the key is seen as 8x16 matrix of bytes – with ai,j,bi,j,i = 0,...,7,j = 0,...,15 we denote the individual bytes of the state and key matrices, respectively.

In each of the 16 rounds, the state S undergoes four byte-oriented transformations, i.e. round R can be represented as:

R = AK ◦ MC ◦ SR ◦ SB

Where AK, MC, SR, SB are acronyms for AddRoundKey, MixColumns, ShiftRows, and SubBytes, respectively. An additional AddRoundKey is perform at the beginning of the state update transformations (known as key prewhitening).

The 1024-bitsubkeyKiused in the i-th round is produced from the previous subkeyKi−1 with similar operations:

Ki =AC◦MC◦SR◦SB(Ki−1)

Where AC stands for AddRound Constant. The prewhitening key K0 is the initial master key. The round and key schedule transformations are the standard operations used in most of the Rijndael-based primitives. For completeness of the description, in the sequel we give a brief definition. The superscripts new, old are used to denote the updated, previous values for the bytes (or the columns).

SubBytes (SB). This transformation is the only non-linear part of the cipher. It consists of independent application of 8x8 bit S-box to all the bytes of the state (or the subkey), etc.

$$a_{i,j}^{new} = S(a_{i,j}^{old}),$$
$$b_{i,j}^{new} = S(b_{i,j}^{old}).$$

We use the invertible AES S-box S(•) for this purpose which is a composition of a finite field inversion and an affine transformation. The precise definition of the S-box is given in Table 1 in the form S(X1X2) = Y.

ShiftRows (SR). It performs a cyclic shift of the rows of the matrix on different offsets that depend on the row index. The value of the offsets ria, rib, i = 0, . . . , 7 is different for the state and the key schedule:

$$a_{i,j}^{new} = a_{i,j+r_a^i}^{old} \bmod 16,$$
$$b_{i,j}^{new} = b_{i,j+r_b^i}^{old} \bmod 16.$$

The precise values are given in Table 2.

MixColumns (MC). The diffusion among the bytes is achieved with this transformation. It is a multiplication of the columns aj,bj of the state/subkeys by a matrix M:

$$a_j^{new} = M \cdot a_j^{old},$$
$$b_j^{new} = M \cdot b_j^{old},$$

Where M is defined as:

$$M = \begin{pmatrix} 02 & 0c & 06 & 08 & 01 & 04 & 01 & 01 \\ 01 & 02 & 0c & 06 & 08 & 01 & 04 & 01 \\ 01 & 01 & 02 & 0c & 06 & 08 & 01 & 04 \\ 04 & 01 & 01 & 02 & 0c & 06 & 08 & 01 \\ 01 & 04 & 01 & 01 & 02 & 0c & 06 & 08 \\ 08 & 01 & 04 & 01 & 01 & 02 & 0c & 06 \\ 06 & 08 & 01 & 04 & 01 & 01 & 02 & 0c \\ 0c & 06 & 08 & 01 & 04 & 01 & 01 & 02 \end{pmatrix}.$$

Table 1: The S-box used in Gear

X_1/X_2	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Table 2: The offsets used in ShiftRows

Row	State offset r_a^i	Key schedule offset r_b^i
0	0	0
1	1	2
2	2	4
3	3	5
4	5	6
5	6	8
6	7	9
7	8	10

We emphasize that the same matrix is used for both the state and key schedule. The

multiplication is performed in GF(28) defined with the irreducible polynomial x8 +x4 +x3 +x+1.

AddRoundKey (AK). The 1024-bitsubkey is xored to the state. The XOR can be seen as byte-wise, i.e.:

$$a_{i,j}^{new} = a_{i,j}^{old} \oplus b_{i,j}$$

AddRoundConstant (AC). A constant Ci is xored to the subkey Ki – in a similar fashion, it can be represented as a byte-wise operation. The value of the constants is dependent on the index i. It is defined as:

$$C_i = \begin{pmatrix} ff & ff & ff & ff & \dots & ff & ff & ff & ff \\ ff & ff & ff & ff & \dots & ff & ff & ff & ff \\ ff & ff & ff & ff & \dots & ff & ff & ff & ff \\ ff & ff & ff & ff & \dots & ff & ff & ff & ff \\ ff & ff & ff & ff & \dots & ff & ff & ff & ff \\ ff & ff & ff & ff & \dots & ff & ff & ff & ff \\ ff & ff & ff & ff & \dots & ff & ff & ff & ff \\ i\oplus 0 & i\oplus 1 & i\oplus 2 & i\oplus 3 & \dots & i\oplus c & i\oplus d & i\oplus e & i\oplus f \end{pmatrix}$$

A. The Hash Function Gear

Once we have defined C-Gear, we use a standard approach to build a hash function based on this cipher. First, we define the compression function CF. It takes two inputs: 1024-bit chaining value Hi and 1024-bit message Mi, and produces 1024-bit chaining value Hi+1 with Davies-Meyer mode of C-Gear, i.e.:

$$H_{i+1} = CF(H_i, M_i) = C\text{-}Gear(H_i, M_i) \oplus H_i$$

Further, we use this compression function to build a hash function with the Merkle-Damgårdconstruction. Briefly, we fix an initial chaining value H0 equal to the first 128 bytes of the fractional part of π (see Table 3). We pad the message M (see below how the padding is performed), and split the expanded message into 1024-bits chunks Mi. Next, we iterate all the message blocks using the compression function based on the Merkle-Damgård construction:

$$H_0 = IV$$
$$H_{i+1} = CF(H_i, M_i)$$

When the expanded message contains l blocks, the output Hl+1 is used to produce the final hash based on truncation, i.e. the hash of M is tr(Hl+1), there tr(X) truncates the leftmost bits of X, depending on the hash size.

Table 3: The initial chaining value H0

24	3F	6A	88	85	A3	08	D3	13
19	8A	2E	03	70	73	44	A4	09
38	22	29	9F	31	D0	08	2E	FA
98	EC	4E	6C	89	45	28	21	E6
38	D0	13	77	BE	54	66	CF	34
E9	0C	6C	C0	AC	29	B7	C9	7C
50	DD	3F	84	D5	B5	B5	47	09
17	92	16	D5	D9	89	79	FB	1B
D1	31	0B	A6	98	DF	B5	AC	2F
FD	72	DB	D0	1A	DF	B7	B8	E1
AF	ED	6A	26	7E	96	BA	7C	90
45	F1	2C	7F	99	24	A1	99	47
B3	91	6C	F7	08	01	F2	E2	85
8E	FC	16	63	69	20	D8	71	57
4E	69							

Thus, for256-bit digests, tr(X) outputs the 256lefttmost (most significant) bits of X, while for 512-bit digest this number is 512. In general, for Gear−n, tr(X) outputs the n most significant bits of the last produced chaining value Hl+1.

The padding. This procedure produces expanded message Me from the original input message M. It assures that the length (in bits) of M is properly encoded into the expanded message Me, and the length of Me is divisible by 1024. To achieve this we use a trivial padding by attaching a required number of 0's to make the last message block1024 bits, and always introduce an addition message block at the end that contains the length of M only.

Let M has t bits. Then from M, first we produce Me˜ = M00...0, where the number of 0's is 1024− (t mod 1024) when t is not divisible by 1024 – otherwise we do not attach any 0's. Next, we attach an additional 1024-bit block that contains 1024 − 64 = 940 zeros, while the last 64 bits are equal to t, i.e. the expanded message is defined as Me = Me˜00...0tbinary.

Endian and mappings. Our hash function is little endian oriented – it regards 64-bit words as 8 bytes in reverse order (with the least significant byte coming first). Furthermore, the mapping of byte sequence to matrix of the state (or the key schedule) is from left to right, and top row to bottom row. For example, the 128-byte sequence a1, . . . , a128 is mapped to the matrix as follows:

$$\begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_{16} \\ a_{17} & a_{18} & a_{19} & \dots & a_{32} \\ \dots & \dots & \dots & \dots & \dots \\ a_{113} & a_{114} & a_{115} & \dots & a_{128} \end{pmatrix}$$

V. PSEUDO CODE AND TEST VECTORS

The pseudo codes of state round, keyschedule round, C-Gear and Gear is given in Algorithm 1-4 respectively

Algorithm 1 State Round(S, Ki)
S ← SubBytes(S)
S ← ShiftRows(S)
S ← MixColumns(S)

S ← AddRoundKey(S, Ki)
end

Algorithm 2 KeySchedule Round(Ki, i)
Ki+1 ← SubBytes(Ki)
Ki+1 ← ShiftRows(Ki+1)
Ki+1 ← MixColumns(Ki+1)
Ki+1 ← AddRoundConstant(Ki+1, i)
end

Algorithm 3 C−Gear(P, K)
S ← AddRoundKey(P, K) K0 ← K
fori = 0 to 15 do
Ki+1 ← KeyScheduleRound(Ki, i)
S ← State Round(S) end for
end

Algorithm 4 Gear(M)
M0|M1|...|Ml ← padded(M) H0 = IV
fori= 0 to l do
Hi+1 =C−Gear(Hi,Mi)⊕Hi
end for
output truncated(Hl)
end
A list of test vectors in given in Table 4.

Table 4: Test vectors for Gear-512

Gear (“ ”)
8798dbba48ffd3b62e239b549499c09b 3d4637273489f9061f5e1d8d214e31ae 1dc13d88a561c5594c9937ee864140e9 7f7b93ffd27e79251d4755a20eca60a4
Gear ("The quick brown fox jumps over the lazy dog")
9b182c6da0010a92e6df1dd67515764b 53a909aecc9be8dbf1c47bf876b4be42 7b96491fbf8e2e90453b4ac9cabf4b5d 73394019ca7801d11307e8d000eed3e2
Gear ("The quick brown fox jumps over the lazy dag")
257269675f2d432ba8dbece0b25d4ac9 a95450c9788a6ef65cee1d1e349b7ed4 a13e0302d0d8204f17832933896ac7e4 4b9709fd6ddb0f86732200955b51648e

VI. CONCLUSION

In this paper, we have presented a new cryptographic hash function Gear that supports digests of up to 512 bits. Our proposal is based on the wide trail strategy and uses an underlying block cipher with 1024 bit key and state. We use mode and construction with longstanding security analysis. Future research might include Hash Function Gear in different constructions as its mode of operation.

REFERENCES

[1] K. Aoki and Y. Sasaki. Preimage attacks on one-block MD4, 63-step MD5 and more. In R. M. Avanzi, L. Keliher, and F. Sica, editors, Selected Areas in Cryptography, volume 5381 of Lecture Notes in Computer Science, pages 103–119. Springer, 2008.

[2] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. On the indifferentiability of the Sponge construction. In N. P. Smart, editor, EURO- CRYPT, volume 4965 of Lecture Notes in Computer Science, pages 181– 197. Springer, 2008.

[3] Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. J. Cryptology, 4(1):3–72, 1991.

[4] A. Biryukov and D. Wagner. Slide attacks. In L. R. Knudsen, editor, FSE, volume 1636 of Lecture Notes in Computer Science, pages 245–259. Springer, 1999.

[5] A. Biryukov and D. Wagner. Advanced slide attacks. In B. Preneel, editor, EUROCRYPT, volume 1807 of Lecture Notes in Computer Science, pages 589–606. Springer, 2000.

[6] J. Black, P. Rogaway, and T. Shrimpton. Black-box analysis of the block- cipher-based hash-function constructions from PGV. In M. Yung, editor, CRYPTO, volume 2442 of Lecture Notes in Computer Science, pages 320– 335. Springer, 2002.

[7] A. Bogdanov, D. Khovratovich, and C. Rechberger. Biclique cryptanalysis of the full AES. In D. H. Lee and X. Wang, editors, ASIACRYPT, volume 7073 of Lecture Notes in Computer Science, pages 344–371. Springer, 2011.

[8] G. Brassard, editor. Advances in Cryptology - CRYPTO ’89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, Au- gust 20-24, 1989, Proceedings, volume 435 of Lecture Notes in Computer Science. Springer, 1990.

[9] A. Canteaut, editor. Fast Software Encryption - 19th International Work- shop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers, volume 7549 of Lecture Notes in Computer Science. Springer, 2012.

[10] J. Daemen, L. R. Knudsen, and V. Rijmen. The block cipher Square. In E. Biham, editor, FSE, volume 1267 of Lecture Notes in Computer Science, pages 149–165. Springer, 1997.

[11] J. Daemen and V. Rijmen. The wide trail design strategy. In B. Honary, editor, IMA Int. Conf., volume 2260 of Lecture Notes in Computer Science, pages 222–238. Springer, 2001.

[12] I. Damg˚ard. A design principle for hash functions. In Brassard [8], pages 416–427.

[13] P. Gauravaram, L. R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schlaffer, and S. S. Thomsen. Grøstl–a sha-3 candidate. Submission to NIST, 2008.

[14] H. Gilbert and T. Peyrin. Super-Sbox cryptanalysis: Improved attacks for AES-like permutations. In Hong and Iwata [15], pages 365–383.

[15] S. Hong and T. Iwata, editors. Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers, volume 6147 of Lecture Notes in Computer Science. Springer, 2010.

[16] J. Jean, M. Naya-Plasencia, and T. Peyrin. Improved rebound attack on the finalist grøstl. In Canteaut [9], pages 110–126.

[17] A. Joux. Multicollisions in iterated hash functions. Application to cascaded constructions. In M. K. Franklin, editor, CRYPTO, volume 3152 of Lecture Notes in Computer Science, pages 306–316. Springer, 2004.

[18] J. Kelsey and T. Kohno. Herding hash functions and the Nostradamus attack. In S.Vaudenay, editor, EUROCRYPT, volume 4004 of Lecture Notes in Computer Science, pages 183–200. Springer, 2006.

[19] J. Kelsey and B. Schneier. Second preimages on n-bit hash functions for much less than 2n work. In R. Cramer, editor, EUROCRYPT, volume 3494 of Lecture Notes in Computer Science, pages 474–490. Springer, 2005.

[20] D. Khovratovich and I. Nikolic. Rotational cryptanalysis of ARX. In Hong and Iwata, pages 333–346.

[21] L. R. Knudsen. Truncated and higher order differentials. In B. Preneel, editor, FSE, volume 1008 of Lecture Notes in Computer Science, pages 196–211. Springer, 1994.
[22] M. Lamberger, F. Mendel, C. Rechberger, V. Rijmen, and M. Schla¨ffer. Rebound distinguishers: Results on the full Whirlpool compression func- tion. In M. Matsui, editor, ASIACRYPT, volume 5912 of Lecture Notes in Computer Science, pages 126–143. Springer, 2009.
[23] S. Lucks. A failure-friendly design principle for hash functions. In B. K. Roy, editor, ASIACRYPT, volume 3788 of Lecture Notes in Computer Science, pages 474–494. Springer, 2005.
[24] M. Matsui. Linear cryptoanalysis method for DES cipher. In T. Helleseth, editor, EUROCRYPT, volume 765 of Lecture Notes in Computer Science, pages 386–397. Springer, 1993.
[25] F. Mendel, C. Rechberger, M. Schla¨ffer, and S. S. Thomsen. The Rebound attack: Cryptanalysis of reduced Whirlpool and Grøstl. In O. Dunkelman, editor, FSE, volume 5665 of Lecture Notes in Computer Science, pages 260–276. Springer, 2009.
[26] R. C. Merkle. One way hash functions and DES. In Brassard [8], pages 428–446.
[27] National Institute of Standards and Technology. Cryptographic hash al- gorithm competition. http://csrc.nist.gov/groups/ST/hash/sha-3/ index.html.
[28] B. Preneel, R. Govaerts, and J. Vandewalle. Hash functions based on block ciphers: A synthetic approach. In D. R. Stinson, editor, CRYPTO, volume 773 of Lecture Notes in Computer Science, pages 368–378. Springer, 1993.
[29] S. Wu, D. Feng, W. Wu, J. Guo, L. Dong, and J. Zou. (Pseudo) preimage attack on round-reduced Grøstl hash function and others. In Canteaut [9], pages 127–145.
[30] Wang, Xiaoyun, Hongbo Yu, and Yiqun Lisa Yin. "Efficient collision search attacks on SHA-0." *Advances in Cryptology–CRYPTO 2005*. Springer Berlin Heidelberg, 2005.
[31] Nandi, M. and S. Paul (2010). "Speeding up the wide-pipe: Secure and fast hashing." Progress in Cryptology-INDOCRYPT 2010: 144-162.
[32] Eli Biham and Orr Dunkelman, "A Framework for Iterative Hash Functions - HAIFA," Cryptology ePrint Archive, 2007. [Online]. http://eprint.iacr.org/2007/278

[33] Alahmad, M. A., I. Al-shaikhli, et al. (2013). “Jouxmulticollisions attack in sponge construction”. The 6th International Conference on Security of Information and Networks (SIN), 2013 6th International Conference on, ACM.

[34] Alahmad, M. A., I. Al-shaikhli, Jumaa, Bashayer (2013). “Protection of the digital Holy Quran PDF file using Combination between AES and RSA Cryptography Algorithms (CARCA)”. Advanced Computer Science Applications and Technologies (ACSAT), 2013 International Conference on, IEEE Xplore.

[35] Alahmad, M. A., I. Al-shaikhli, Duwaikh, Amal (2013). “A New Fragile Digital Watermarking Technique for a PDF digital Holy Quran”. Advanced Computer Science Applications and Technologies (ACSAT), 2013 International Conference on, IEEE Xplore.