Joux multicollisions attack in sponge construction

Mohammad A. AlAhmad
Department of Computer Science, International Islamic University of Malaysia, 53100 Jalan Gombak
Kuala Lumpur, Malaysia
malahmads@yahoo.com

Imad Fakhri Alshaikhli
Department of Computer Science, International Islamic University of Malaysia, 53100 Jalan Gombak
Kuala Lumpur, Malaysia
imadf@iium.edu.my

Mridul Nandi
Applied Statistics Unit, Indian Statistical Institute,
Kolkata, India
mridul_r@isical.ac.in

ABSTRACT

Cryptographic hash functions take an unfixed size of input and produce a fixed size of an output. A hash function usually has two main components: a compression function and mode of operation. Sponge construction is one of the main operations of modes of used in modern cryptographic hash function. In this paper, we present multicollisions attack in sponge construction. In 2004, Joux [3] presented multicollision attack in iterated hash function. Our attack is similar to Joux attack but specifically for sponge construction[1]. We show that finding multicollisions in sponge construction of messages that hash to the same value, is not harder finding ordinary collisions. Then, we use this attack as a tool to prove that concatenating more than one hash function in order to increase the security level does not yield to more secure construction.

Categories and Subject Descriptors

K.6.5 [**MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS**]: Security and Protection – *Authentication, Insurance.*

General Terms

Security

Keywords: Multicollisions - Sponge - Concatenation

1. INTRODUCTION

Cryptographic hash functions take an unfixed size of input and produce a fixed size of an output. A hash function usually has two main components: a compression function and mode of operation. The mode of operation is the design of the hash function that iterates the basic compression function several times. But good hash function should behave like a random oracle [1].Therein, a random oracle maps a variable input message to infinite uniformly and independent distributed output message. It is completely random, for each possible input x, there is a completely random value $h(x)$. In real life, a true random oracle doesn't exist, but an ideal designed hash function should inherit the security criteria of this theoretical random oracle construction, because, it is unreachable goal for an iterated hash functions to behave exactly and become strong as random oracles.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIN '13, November 26 - 28 2013, Aksaray, Turkey
Copyright 2013 ACM 978-1-4503-2498-4/13/11 ...$15.00.
http://dx.doi.org/10.1145/2523514.2523551

Practically, the main security requirement for hash functions is collision resistance. Collision resistance is finding two different messages M and M' that leads to the same hash digest. According to the birthday paradox theory, in a group of 23 randomly chosen people, two people will share a birthday with probability at least ½. More precisely, suppose h denotes the domain set for all human beings, and for all x, $h(x)$ denotes the birthday of person x. Then, the range of h consists of 366 days (if we include February 29). So that, finding two people with the same birthday is the same thing as finding a collision for this particular hash function [2]. For this reason, a hash function for which collision cannot be efficiently solved is often said to be collision resistant [2]. Accordingly, if h is a

hash function that output bit values, then among the hash values of $2^{n/2}$ different messages, there exists a collision of birthday paradox based attack. The birthday attack imposes a lower bound on the sizes of secure hash digest. Yet, a 64-bit message hash digest considered unsecure. A collision can be found with probability ½ with just over 2^{32} random hashes. As a result, the minimum acceptable message hash digest today is 160 bits or larger due to the birthday attack. The complexity of the birthday attack is $\Theta(2^{n/2})$. Also, the birthday attack is applicable to all compression functions in iterated hash functions.

2. PRELIMINARIES

In this section, we give brief introductions about Joux attack and sponge construction. These two main concepts are the basis for the attack presented in this paper.

2.1 Joux Attack

Recently, Joux [3] showed an effective multicollisions attack in iterated hash function and it's faster than birthday attack. But before describing Joux attack which is the basis for the attacks used in this paper, the hashing process of the iterated hash functions; i.e. Merkle-Damgård; will be described as follows:

- Break the input x into blocks $x_1, x_2.....x_t$.
- Pad the last block x_t with 0-bits if necessary to obtain the multiple length of r.
- Create the length block x_{t+1} with bit length r to hold the right justified binary representation of overall bit-length of x (MD

strengthen).

- Inputting x_1, x_2.....x_t to the compression function (iterated processing) to produce an intermediate value of H_i.
- H_i serves as feedback value to f and is processed with x_{i+1} in the next iteration. This implies the need of an initial value *(IV)* H_0 for the first iteration that is often provided pre-defined with bit-length *r*.
- Output $H(x_t)=H_t$. [7]

The most distinctive and special part of Merkle-Damgård construction (mode of operation) is that the problem of designing a collision resistant hash function reduced to designing a collision resistant compression function. This means, if the compression function is collision resistant, then, the hash function is collision resistant. So, the properties of the compression function will be transformed to the hash function. Joux found that finding multicollisions, i.e. *r*-tuples of messages that all hash to the same values, is not harder than finding ordinary collisions, i.e. pairs of messages, even for extremely large values of *r*. Then, he uses his multicollisions as a tool to prove that concatenating the results of several independent iterated hash functions, i.e. Merkle-Damgård, in order to build a larger one, does not yield to a secure construction [3]. To describe such attack, the padding process is ignored as long as the messages collisions have the same length. This means, collision with padding will yield to the collision without padding. An adversary can access a collision finding machine *C,* that given as input as a chaining value *h* outputs two different blocks *X* and *X'* such that $f(h,X) = f(h,X')$. The collision finding machine *C* should work for all chaining values of the compression function. To illustrate the basic idea, he showed how is the 4-collisions can be obtained with two calls to *C.* Starting from the initial value *IV,* an adversary may call *C* to find the first two different blocks X_0 and X_0' that yield to a collision where $f(h,X_0) = f(h,X_0')$. Again, the same process will run again but with different blocks X_1 and X_1' such that $f(h,X_1) = f(h,X_1')$. Putting the 2 processes altogether, we obtain the following 4-collisions:

$$f(f(IV,X_0), X_1) = f(f(IV,X_0), X_1') = f(f(IV,X_0'), X_1) = f(f(IV,X_0'), X_1').$$

Now, this idea can be extended to find 2^t collisions in *h(x)* and can concluded as follows:

- Let $h_0 = IV$
- For *i* from 1 to *t* do:
  - Call *C* and find X_i and X_i' where $f(h_{i-1},X_i) = f(h_{i-1},X_i')$
  - Let $h_i = f(h_{i-1},X_i)$
- Pad and output the 2^t messages of the form (x_1,……., x_t, Padding) where x_i is one of the two blocks X_i and X_i' [3].

As Figure 1 shows that all the intermediate hash values are equal since all of the 2^t hashing process go through h_0, h_1,….., h_t [3].

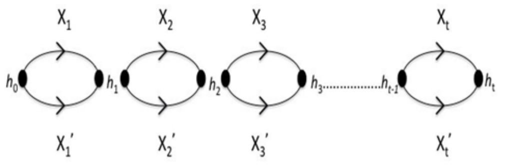

Figure1. Schematic representation of multicollision construction [3]

Joux used multicollision attack as a tool to produce a collision attack in concatenated hash functions *F* and *G*. He claimed that concatenating two different iterated hash functions *(F(X)IIG(X))* or slightly different ones is not really secure than *F* or *G* itself. This result is applied to collision resistance, preimage and second preimage resistance. For a collision resistance, if the security level of *F* is $2^{nf/2}$ and for *G* is $2^{ng/2}$, then the complexity of the best attack for *FIIG* would be $2^{(nf+ng)/2}$. But Joux had much better attack which find collisions on *FIIG* with complexity $n_g 2^{nf/2}+2^{ng/2}$ if $n_f \leq n_g$. To describe such attack, first an adversary may use the multicollisions attack showed in Figure 1 with *t* equal to $n_g/2$ rounded up, to construct a 2^t collision on *F* and this costs about $t2^{nf/2}$ operations in the compression function *f.* This yields to 2^t different messages of hash values on the *F* side. Then, perform a direct application of the birthday paradox on the obtained set among the n_g-bit hashes of these 2^t messages by *G*. This attack does not require of *G* to be an iterative hash function, any hash function will do. Also, for preimage and second preimage resistance, assume that an adversary is hashing messages for a relatively small set of 2^m messages. Clearly, the best generic attack is to find a preimage in that case which costs 2^m steps. And assuming that the output of each of the two concatenated hash functions is larger than *m* bit and on this set of messages, i.e. *F*, hash a shortcut attack [3]. Then, uses this attack to recover a candidate preimage or second preimage. So, this indicates the preimage or second preimage for the other function. This attack shows the same result as the collision attack which concatenating two different iterated hash functions *(F(X)IIG(X))* or slightly different ones is not really secure than *F* or *G* itself. More particularly, this attack much better attack than collision resistance where its complexity is $n_g 2^{nf/2}+2^{ng/2}+2^{nf}$ if $n_f \geq n_g$. To describe such attack, first an adversary may use the multicollisions attack showed in Figure 1 with *t* equal to n_g rounded up, to construct a 2^t collision on *F* and this costs about $t2^{nf/2}$ operations in the compression function *f.* Then, search for an additional block (including the padding of the message) that maps the last chaining value to the target value of *F*. Lastly, the adversary will obtain 2^t different messages with the expected hash value on the *F* side. As before, preimage and second preimage attack does not require of *G* to be an iterative hash function, any hash function will do. Indeed, Joux attack had a direct impact on Merkle-Damgård construction and made the cryptography community to look forward for a new trusted construction. Consequently, Stefan lucks [4] introduced the wide pipe hash construction as an intermediate version of Merkle-Damgård to improve the structural weaknesses of Merkle-Damgård design. Figure 2 shows the wide pipe hash construction. 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.

[1]This is not the case for Keccak SHA-3 winner candidate, where Keccak is a wide pipe sponge construction which has the capacity $c = 2n$, and the hash digest is *n*, which means, there is a truncation process in the last function *f*, hence the attack in this paper is not applicable for Keccak hash function.

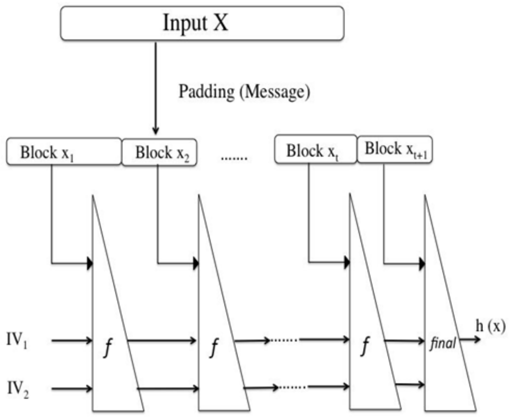

Figure 2. The wide pipe hash construction [4]

Also, the final compression function compresses the internal state length (for ex, *2n-* bit) to output a hash digest of *n*-bit. This simply can be achieved by discarding the last half of *2n*-bit output. Also, Mridul Nandi and Souradyauti Paul [5] proposed the fast wide pipe construction. It is twice faster than the wide pipe construction. Figure 3 shows the fast wide pipe construction. As the Figure shows, the input (*IV*s) for each compression function is divided into halves.

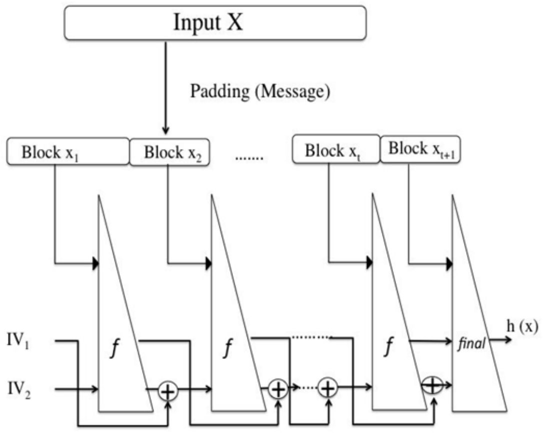

Figure 3. The fast wide pipe hash construction [5]

The first half is inputted in the compression function and the other half is XORed with the output for the same compression function. The feed-forward process makes the overall design faster. Hence, faster process is obtained. The final output of the hash digest can be truncated to the desired digest length using the final compression function.

2.2 The Sponge Construction

Sponge construction is an iterative construction designed by Guido Bertoni, Joan Daemen, Micheal Peeter and Gilles Van Assche to replace Merkle-Damgård construction [6]. It is a construction that maps a variable length input *"M"* to a variable length output. Namely, by using a fixed-length transformation (or permutation) *f* that operates on a fixed number of $b = r + c$ bits. Where *b* is the width, *r* is the bitrate, *c* is the capacity and variable output called *Z* of length *n* as Figure 4 shown. The sponge construction operates in three phases:

- ***Initialization:*** the message *"M"* is padded by appending a '1' bit followed by the minimal (possibly zero) number of '0' bits to reach a length that is a multiple of *r*.
- ***Absorbing phase:*** The *r*-bit message blocks are XORed with the first *r* bits of the state of the function *F*. After processing all the message blocks, the squeezing phase starts.
- ***Squeezing phase:*** The first *r* bits of the state are returned as output blocks of the function *F*. lastly, the number of output blocks is chosen by the user [6].

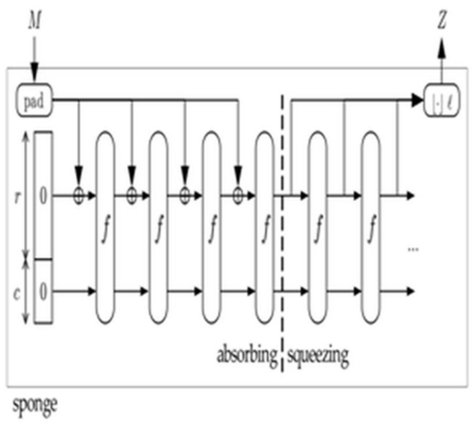

Figure4. The sponge construction [6]

3. CONSTRUCTING MULTICOLLISIONS

In this section, we show that multicollisions attack on sponge construction can be done in an efficient way. But before describing the attack, let us remark that the padding process is neglected as long as the messages have the same length. More precisely, since the padding process occur in the last block of message *"M",* then, all intermediate chaining values have identical lengths. Also, presenting the notion capacity which was introduced first in sponge construction et. al [6]. It was the first step towards separating the hash digest length from security level of hash functions. So that, it is clear that the smaller capacity *c*, the more vulnerable sponge construction become. Where the original Merkle-Damgård construction assumes that *c=n,* which expose it to Joux attack. A possible direction for Merkle-Damgård construction is to enlarge the internal state, i.e. *2n*, and then truncates the final desired hash digest output with transformation function, i.e. *n,* as shown in the patched versions of Merkle-Damgård construction in the section 2.1. Clearly, the security level of a hash function is limited to the length of the hash digest of that hash function. Where, enlarging the internal state, i.e. *2n*, then, truncating the final result to *n*, resists Joux attack of that hash function. More precisely, in sponge construction, the resistance of inner collision is limited with capacity *c* with complexity of the order $2^{c/2}$. To describe the multicollision attack in sponge construction, we assume that the capacity *c=n*, where *n* is the size of the hash digest. And, we assume that we can access the finding collision machine *C,* where given as input *h* outputs two different blocks *Z* and Z' such that $f(h, Z) = f(h, Z')$. This collision finding machine will use the generic birthday attack to find the collision. To illustrate the basic idea, we first show how 4-collisions can be obtained with two calls to *C*. Staring from the initial value *IV,* we use a first call to *C* to obtain two different blocks Z_0 and Z_0' that yield to collision $f(h, Z_0) = f(h, Z_0')$. Let d_0 denotes this common value and using a second call to *C*, we will find two other blocks Z_1 and Z_1' such that $f(h, Z_1) = f(h, Z_1')$ [3]. Putting these two steps together, we obtain the following 4-collisions:

$$f(f(IV, Z_0), Z_1) = f(f(IV, Z_0), Z_1') = f(f(IV, Z_0'), Z_1) = f(f(IV, Z_0'), Z_1')$$

In more details to describe the attack, we will perform the generic birthday attack 1 (*BD1)* for messages M_1 and M_1' (assuming W_1=0), then input them to *f* along with h_0. The output of finding collision machine *C* are Z_1 and Z_1' where $f(h, Z_1) = f(h, Z_1') = d_0$ (collision of birthday attack 1). For the next iteration of *f*, we will adjust the value of Z_1 and Z_1' to become 0. That's will output W_2 and h_1 which are the inputs for the next iteration along with the generic birthday attack 2 (*BD2)* for messages M_2 and M_2'. The output of finding collision machine *C* are Z_2 and Z_2' where $f(h, Z_2) = f(h, Z_2') = d_1$ (collision of birthday attack 2). Figure 5 shows the basic idea of the collision.

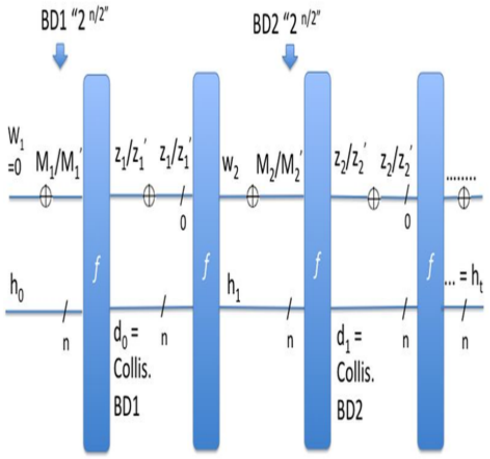

Figure 5. Multicollisions in sponge construction

This basic idea can be extended to much larger collisions by using more calls to machine *C*. More precisely, using *t* calls, we can build 2^t in *H* [3]. The attack work as follows:

- Let W_1=0 XORed with M_1 and M_1'.
- Let $h_0 = IV$.
- For *i* from 1 to *t* do:

1. Call *C* and find Z_i, Z_i' and d_{i-1} where $f(h_{i-1}, M_i) = Z_i$ and $f(h_{i-1}, M_i') = Zi'$.
2. And, $d_{i-1} = f(h_{i-1}, M_i) = f(h_{i-1}, M_i') = Z_i = Z_i'$.
3. Let Z_i, Z_i' = 0 and call *C* to produce $h_i = f(d_{i-1}, Z_i/Z_i')$.
4. Let W_i XORed with M_i/M_i' then go to step 1, we repeat this process until we obtain the final result h_t.

- Pad and output the 2^t messages of the form (m_1,……., m_t, Padding) where m_i is one of the two blocks M_i and M_i'.

Figure 6 shows a schematic representation of multicollisions on sponge construction which generalize the idea of the attack. The collision d_0 is obtained every other iteration (or permutation). For example, we need 4 iterations (or permutations) of *f* in order to obtain 2-collisions. This yields to have 4 different pairs of combinations, i.e. M_1IIZ_1, M_1IIZ_1', M_1'IIZ_1 and M_1'IIZ_1'. So that, this particular attack costs $2*2^{n/2}$ with complexity $\Theta\ (2^{n/2} + \frac{n}{2})$.

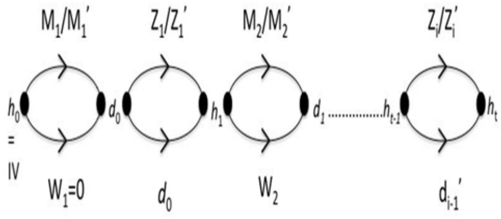

Figure 6. Schematic representation of multicollisions on sponge construction

Generalizing the attack, we obtain a cost with $(\frac{n}{2})*2^{n/2}$ with complexity $\Theta\ ((\frac{n}{2})*2^{n/2} + \frac{n}{2})$.

4. ON THE SECURITY OF THE CONCATENATED HASH FUNCTIONS

Joux proved that concatenating two hash functions (at least one of them has MD construction), i.e. *F*II*G*, is not really secure than *F* or *G* itself. This result is applicable to collision resistance, preimage resistance and second preimage resistance. In this section, we prove that concatenating two hash functions, i.e. *F*II*G*, where at least one of them has sponge construction is not really secure than *F* or *G* itself. We apply this result to collision resistance, preimage resistance and second preimage resistance.

4.1 Collision resistance

With respect to collision resistance, the security level of any hash function is $2^{n/2}$. Then, if the security level of *F* is $2^{nf/2}$ and for *G* is $2^{ng/2}$, then the complexity of the best attack for *FIIG* would be $2^{(nf+ng)/2}$. But there exists much better attack which find collisions for *FIIG* with complexity $n_g 2^{nf/2} + 2^{ng/2}$ if $n_f \leq n_g$. To describe such attack, we use the multicollision algorithm explained in section 3 as a tool with *t* equal to $n_g/2$ rounded up, to construct a 2^t collision on *F* side and this costs about $t2^{nf/2}$ operations in the compression function *f*. This yields to 2^t different messages of hash values on the *F* side. Then, perform a direct application of the birthday paradox on the obtained set among the n_g-bit hashes of these 2^t messages by *G*. This attack does not require of *G* to be an iterative hash function, any hash function will do (but, one of them require having sponge construction). Thus, concatenating two independent hash functions does not improve the collision resistance [3].

4.2 Preimage and second preimage resistance

With respect to preimage and second preimage resistance, the security level of any hash function is 2^n. Then, if the security level of *F* is 2^{nf} and for *G* is 2^{ng}, then the complexity of the best attack for *FIIG* would be $2^{(nf+ng)}$. As with collision resistance, there exists much better attack which find collisions for *FIIG* with complexity $n_g 2^{nf/2} + 2^{ng/2} + 2^{nf}$ if $n_f \geq n_g$. To describe such attack, we use the multicollision algorithm explained in section 3 as a tool of Figure 1 with *t* equal to n_g rounded up, to construct a 2^t collision on *F* and this costs about $t2^{nf/2}$ operations in the compression function *f*. Then, search for an additional block (including the padding of the message) that maps the last chaining value to the target value of *F*. Lastly, the adversary will obtain 2^t different messages with the expected hash value on the *F* side. Once again, preimage and second preimage attack does not require of *G* to be an iterative hash function, any hash function will do [3] (but, one of them require having sponge construction). Thus, concatenating two independent hash functions does not improve the preimage and second preimage resistance.

5. EXTENSIONS OF MULTICOLLISION ATTACK

Multicollisions attack presented in section 3 can be extended to three or more concatenated hash functions. To illustrate the idea, we assume K is a third hash function with security level 2^{nk}, then, we use the attack in section 3 to build a 2^t collision on *FIIG*. This yield to collision on K based on Joux observation. When $n_f = n_g = n_k = n$, the expression of the complexity is simplified to $n^2 . 2^{n/2}$. In general, a simultaneous collision on s different n-bit hash functions can be found with complexity $n^{s-1}.2^{n/2}$. This means, the security level of such a construction stays within the security of a single hash function. Also, if we build a hash function by concatenating *G(F(X)IIX)* or *(G(X)IIF(X)))*. This is more complicated than the *GIIF* construction. But, the same attack can be applied, where a 2^t is collisions is found on *F(X)*, which fixes the *F(X)* in the first half of the big hash function, also the copy of *F(X)* in the call to *G* [3], which yields to collision on the part of *G*. The preimage attack can be adapted to two different concatenated examples presented here of the concatenated hash functions.

6. CONCLUSION

In this paper, we have showed that multicollisions attack on sponge construction is not really harder than ordinary collision. Also, this attack is used to prove that concatenating two separate independent hash functions is not secure than using one of them by itself. This result is applied to collision resistance, preimage resistance and second preimage resistance. Modern hash functions designers that use sponge construction should consider our attack by enlarging the capacity c at least double the size of the hash digest to obtain excellent resistance.

7. REFERENCES

[1] Bellare, M., & Rogaway, P. 1993. Random oracles are practical: A paradigm for designing efficient protocols. *Paper presented at the Proceedings of the 1st ACM conference on Computer and communications security.*

[2] Stinson, D. R. 2006. Cryptography: theory and practice: *CRC press.*

[3] Joux, A. 2004. Multicollisions in iterated hash functions. Application to cascaded constructions. *Paper presented at the Advances in Cryptology–CRYPTO 2004.*

[4] Lucks, S. 2004. Design principles for iterated hash functions, *Cryptology ePrint Archive,* Report 2004/253. DOI = http://eprint. iacr. org

[5] Nandi, M. and S. Paul 2010. Speeding up the wide-pipe: Secure and fast hashing. *Paper presented in Cryptology-INDOCRYPT 2010*: 144-162.

[6] Bertoni, G., J. Daemen 2007. Sponge functions. *ECRYPT hash workshop*.

[7] Damgård, I. B. 1990. A design principle for hash functions. *Advances in Cryptology—CRYPTO'89 Proceedings*, Springer.

[8] Taha, I. a. A., Mohammad and Munther, Khansaa 2012. Comparison and analysis study of sha-3 finallists. *International Conference on Advanced Computer Science Applications and Technologies*(26-28 Nov 2012), 7.

[9] Biham, E., R. Chen 2005. Collisions of SHA-0 and Reduced SHA-1. *Advances in Cryptology–EUROCRYPT*, Springer: 36-57.

[10] Matusiewicz, K. and J. Pieprzyk 2006. Finding good differential patterns for attacks on SHA-1. *Coding and Cryptography*, Springer: 164-177.

[11] Biham, E. and O. Dunkelman 2006. A framework for iterative hash functions-HAIFA. *Second NIST Cryptographic Hash Workshop.*

[12] Bertoni, G., J. Daemen 2009. "Keccak specifications." *Submission to NIST (Round 2).*