Master's Project Proposal

Soham Sadhu
Department of Computer Science
Rochester Institute of Technology
Rochester, NY 14623
sxs9174@rit.edu

July 14, 2013

Chair:	Prof. Stanisław Radziszowski	spr@cs.rit.edu

signature date

Reader:	Prof. Alan Kaminsky	ark@cs.rit.edu

signature date

Observer:	Prof. Edith Hemaspaandra	eh@cs.rit.edu

signature date

Abstract

Hash functions, have applications in computer security, in fields of authentication and integrity. Due to importance of hash function usage in everyday computing, standards for using hashing algorithm and their bit size have been released by (NIST) which are denoted by nomenclature Standard Hashing Algorithm (SHA).

Due to advances in cryptanalysis of SHA-2, NIST announced a competition in November, 2007 to choose SHA-3. In October, 2012 the winner was selected to be Keccak amongst 64 submissions. All the submissions were open to public scrutiny, and underwent intensive third party cryptanalysis, before the winner was selected. Keccak was chosen for its flexibility, efficient and elegant implementation, and large security margin.

All algorithms submitted to competition have undergone public scrutiny. And other four finalist in the competition were almost equivalent to Keccak, in attributes of security margin and implementation. In this project, I will be comparing Keccak with two other SHA-3 finalists, BLAKE, and Grøstl with respect to their resistance to simulated annealing and tabu search.

Application of tabu search and simulated annealing to hash algorithms will be akin to generic attacks. That is these methods of breaking hash functions are design agnostic or do not depend on the workings of the hash function. Thus ensuring no bias in the experiment. At present, it is computationally infeasible to break the above mentioned hash functions. But the reduced versions of these can be subjected to attacks for near collisions. Thus I will be able to examine and conclude, if reduced instance Keccak has better resistance to generic attacks than reduced instance of BLAKE and Grøstl.

1 Problem Statement

1.1 Hash Functions

A cryptographic hash function, is an algorithm capable of intaking arbitrarily long input string, and output a fixed size string, often as called message digest. The message digest for two strings even differing by a single bit should ideally be completely different, and no two input message should have the same hash value. This property enables us to finger print a message. Following are the properties of and ideal hash function [5].

1. **Preimage resistance**

PREIMAGE
Given: A hash function $h : \mathcal{X} \to \mathcal{Y}$ and an element $y \in \mathcal{Y}$.
Find: $x \in \mathcal{X}$ such that $h(x) = y$.

If the preimage problem for a hash function cannot be efficiently solved, then it is preimage resistant. That is the hash function is one way, or rather it is difficult to find the input, given the output alone.

2. **Second preimage resistance**

SECOND PREIMAGE
Given: A hash function $h : \mathcal{X} \to \mathcal{Y}$ and an element $x \in \mathcal{X}$.
Find: $x' \in \mathcal{X}$ such that $x' \neq x$ and $h(x) = h(x')$.

A hash function for which a different input given another input, that compute to same hash cannot be found easily, is called as having second preimage resistance.

3. **Collision resistance**

COLLISION
Given: A hash function $h : \mathcal{X} \to \mathcal{Y}$ **Find:** $x, x' \in \mathcal{X}$ such that $x' \neq x$ and $h(x') = h(x)$.

Collision problem states that, can two different input strings be found, such that they hash to the same value given the same hash function. A hash function is collision resistant, if it is computationally infeasible to find two different values hashing to same value.

1.2 Standards and NIST Competition

Since hash functions can finger print any data, they find wide applications in computer security. And thus there needs to be a standard for implemenation and application of hash function, which is provided by National Institute of Standards and Technology(NIST). SHA-0 was initially proposed by National Security

Agency(NSA) as a standardised hashing algorithm in 1993. It was later standardised by NIST. In 1995 SHA-0 was replaced by SHA-1 designed by NSA [3,4]. SHA-2 was designed by NSA, and released in 2001 by NIST. It is basically a family of hash functions consisting of SHA-224, SHA-256, SHA-384, SHA-512. The number suffix after the SHA acronym, indicates the bit length, of the output of that hash function. Although SHA-2 family of algorithms were influenced by SHA-1 design, but the attacks on SHA-1 have not been successfully extended completely to SHA-2.

In response to advances made in cryptanalysis of SHA-2. NIST announced a public competition on November, 2007; for a new cryptographic hash algorithm, that would be SHA-3. 51 candidates from 64 submissions for first round of competition were announced in December, 2008. In October, 2012 NIST announced the winner of the competition to be Keccak, amongst the other four finalist, which were BLAKE, Grøstl, JH and Skein. Keccak was chosen for its' large security margin, efficient hardware implementation, and flexibility.

1.3 Motivation

The arguments for choosing Keccak as SHA-3 are strong. However, other 4 finalists, have equally strong claim to security margin; one of the attributes on which Keccak was chosen. All the finalists have gone public scrutiny, and have shown resistance, to a number of attacks.

HYPOTHESIS
Reduced round Keccak, will have better resistance to near collisions found by tabu search and simulated annnealing, compared to reduced round BLAKE and Grøstl.

The aim of the project is to check if reduced version of Keccak holds security margin comparable, to reduced versions of BLAKE and Grøstl. This will be done by subjecting the reduced versions of the 3 algorithms to, a search in chaining value for two different messages. The search will be done in k-bit neighbourhood of initial chaining value using tabu search and simulated annealing. This should lead to chaining values that can give possible near collisions. These experiments would not prove that the said algorithms are vulnerable to attacks, but would rather provide a security margin on which they can be compared.

2 Background

2.1 Grøstl

Grøstl is collection of hash functions which produce digest size, ranging from 1 to 64 bytes. The variant of Grøstl that returns a message digest of size n, is called Grøstl-n. Grøstl is an iterated hash function, with two compression functions named P and Q. The input is padded and then split into l-bit message blocks m_1,..., m_t, and each message block is processed sequentially. The initial l-bit chaining value h_0 = iv is defined, and the blocks m_i are processed as $h_i \leftarrow f(h_{i-1}, m_i)$ for i = 1,..., t. For variants up to 256 bits output, size of l is 256 bits. And for digest sizes larger than 256 bits, l is 1024 bits. After the last message block is processed, the last chaining value output is sent through a Ω function, to get the hash output H(M) [6].

$H(M) = \Omega(h_t),$

The f function shown above, is composed of two l-bit permutations called P and Q, which is defined as follows.

$f(h, m) = P(h \oplus m) \oplus Q(m) \oplus h.$

The Ω function consists of a $trunc_n(x)$ that outputs only the trailing n bits of input x. $\Omega(x) = trunc_n(P(x) \oplus x)$.

In order to fit the varying input length message to the block sizes of l padding is defined. First bit '1' is appended, then $w = -N - 65 modl$ 0 bits are appended; where N is the length of the original message. Finally a 64 bit representation of $(N + w + 65)/l$.

There are two variations for P and Q permutations, one each for the digest size lower and higher than 256 bits. There are four round transformations, that compose a round R. The permutation consists of a number of rounds R, and can be represented as

$R = MixBytes \cdot ShiftBytes \cdot SubBytes \cdot AddRoundConstant$

The transformations SubBytes and MixBytes are same for all transformation while, ShiftBytes and AddRoundConstant differ for each of the transformations. The transformations operate on matrix of bytes, with the permutation of lower size digest having matrix of 8 rows and 8 columns, while that for larger variant is of 16 columns and 8 rows. The number of rounds for digest sizes upto 256 bits are 10. For digest sizes higher than that, number of rounds are 14. The individual components of each round for P and Q are described below.

AddRoundConstant: transformation round XOR a round dependant constant to the state matrix say A. It is represented as $A \leftarrow A \oplus C[i]$, where C[i] is the round

constant in round i.

SubBytes: substitutes each byte in state by value from S-box. Say $a_{i,j}$ a element in row i and column j of the state matrix, then the transformation done is $a_{i,j} \leftarrow S(a_{i,j}), 0 \leq i < 8, 0 \leq j < v$.

ShiftBytes: transformation cyclically shifts the bytes in a row to left by that number. Let list vector of a number denote the shift, with the index of the element indicating the row. The vector representation for P_{512} = [0, 1, 2, 3, 4, 5, 6, 7] and Q_{512} = [1, 3, 5, 7, 0, 2, 4, 6]. Those for the larger permutation are P_{1024} = [0, 1, 2, 3, 4, 5, 6, 11]and Q_{1024} = [1, 3, 5, 11, 0, 2, 4, 6].

MixBytes: transformation, multiplies each column of the state matrix A, by a constant 8 $\times$ 8 matrix B. The transformation, can be shown as $A \leftarrow B \times A$. The matrix B, can be seen as a finite field over $\mathbb{F}_{256}$. This finite field is defined over $\mathbb{F}_2$ by the irreducible polynomial $x^8 \oplus x^4 \oplus x^3 \oplus x \oplus 1$.

2.2 BLAKE

BLAKE [1] hash function is built on HAIFA (HAsh Iterative FrAmework) structure [2] which is an improved version of Merkle-Damg$\grave{a}$rd function. BLAKE has 4 variations of the algorithm that can give only 4 different digest lengths. The construction takes in 4 inputs, one message; two a salt, that makes function that parameter specific; and three a counter, which is count of all the bits hashed till then; and lastly a chaining value which is input of the previous operation or initial value in case of hash initiation. The compression function is composed of a 4 $\times$ 4 matrix of words. Where one word is equal to 32 bits for BLAKE-256 variant, while 64 bit for variant BLAKE-512.

Symbol	Meaning
$\leftarrow$	variable assignment
$+$	addition modulo 2^{32} or (modulo 2^{64})
$\gg k$	rotate k bits to least significant bits
$\ll k$	rotate k bits to most significant bits
$\langle l \rangle_k$	encoding of integer l over k bits

Table 1: Convention of symbols used in BLAKE algorithm

2.2.1 BLAKE-256

The compression function takes following as input

- a chaining value of $h = h_0, \ldots, h_7$
- a message block $m = m_0, \ldots, m_{15}$
- a salt $s = s_0, \ldots, s_3$

- a counter $t = t_0, t_1$

These four inputs of 30 words or 120 bytes, are processed as $h' = compress(h, m, s, t)$ to provide a new chain value of 8 words.

Compression function

- **Constants**

IV_0 = 6A09E667	IV_1 = BB67AE85	IV_2 = 3C6EF372	IV_3 = A54FF53A
IV_4 = 510E527F	IV_5 = 9B05688C	IV_6 = 1F83D9AB	IV_7 = 5BE0CD19

Table 2: Initial values which become the chaining value for the first message block

c_0 = 243F6A88	c_1 = 85A308D3	c_2 = 13198A2E	c_3 = 03707344
c_4 = A4093822	c_5 = 299F31D0	c_6 = 082EFA98	c_7 = EC4E6C89
c_8 = 452821E6	c_9 = 38D01377	c_{10} = BE5466CF	c_{11} = 34E90C6C
c_{12} = C0AC29B7	c_{13} = C97C50DD	c_{14} = B5470917	c_{15} = 3F84D5B5

Table 3: 16 constants used for BLAKE-256

- **Initialization:** The constants mentioned are used with the salts, and counter along with initial value used as chaining input, to create a initial matrix of 4×4, 16 word state.

$$\begin{pmatrix} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{pmatrix} \leftarrow \begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ s_0 \oplus c_0 & s_1 \oplus c_1 & s_2 \oplus c_2 & s_3 \oplus c_3 \\ t_0 \oplus c_4 & t_0 \oplus c_5 & t_1 \oplus c_6 & t_1 \oplus c_7 \end{pmatrix}$$

- **Round function:** After initialisation, the state is subjected to column and diagonal operations, 14 times. A round operation G acts as per following where the round function $G_i(a, b, c, d)$ sets

$a \leftarrow a + b + (m_{\sigma_r(21)} \oplus c_{\sigma_r(2i+1)})$

$d \leftarrow (d \oplus a) \gg 16$

$c \leftarrow c + d$

$b \leftarrow (b \oplus c) \gg 12$

σ_0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
σ_1	14	10	4	8	9	15	13	6	1	12	0	2	11	7	5	3
σ_2	11	8	12	0	5	2	15	13	10	14	3	6	7	1	9	4
σ_3	7	9	3	1	13	12	11	14	2	6	5	10	4	0	15	8
σ_4	9	0	5	7	2	4	10	15	14	1	11	12	6	8	3	13
σ_5	2	12	6	10	0	11	8	3	4	13	7	5	15	14	1	9
σ_6	12	5	1	15	14	13	4	10	0	7	6	3	9	2	8	11
σ_7	13	11	7	14	12	1	3	9	5	0	15	4	8	6	2	10
σ_8	6	15	14	9	11	3	0	8	12	2	13	7	1	4	10	5
σ_9	10	2	8	4	7	6	1	5	15	11	9	14	3	12	13	0

Table 4: Round permutations to be used

$$G_0(v_0, v_8, v_{12}) \quad G_1(v_1, v_5, v_9, v_{13}) \quad G_2(v_2, v_6, v_{10}, v_{14}) \quad G_3(v_3, v_7, v_{11}, v_{15})$$
$$G_4(v_0, v_5, v_{10}, v_{15}) \quad G_5(v_1, v_6, v_{11}, v_{12}) \quad G_6(v_2, v_7, v_8, v_{13}) \quad G_7(v_3, v_4, v_9, v_{14})$$

$a \leftarrow a + b + (m_{\sigma_r(2i+1)} \oplus c_{\sigma_r(2i)})$
$d \leftarrow (d \oplus a) \gg 8$
$c \leftarrow c + d$
$b \leftarrow (b \oplus c) \gg 7$
The implementation of the G function is shown below.

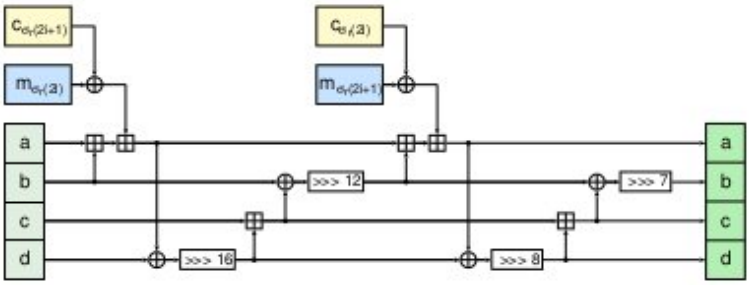

Figure 1: The G_i function in BLAKE

- **Finalization:** The chaining values for the next stage are obtained by XOR of the words from the state matrix, the salt and the initial value.
  $h'_0 \leftarrow h_0 \oplus s_0 \oplus v_0 \oplus v_8$
  $h'_1 \leftarrow h_1 \oplus s_1 \oplus v_1 \oplus v_9$
  $h'_2 \leftarrow h_2 \oplus s_2 \oplus v_2 \oplus v_{10}$
  $h'_3 \leftarrow h_3 \oplus s_3 \oplus v_3 \oplus v_{11}$
  $h'_4 \leftarrow h_4 \oplus s_0 \oplus v_4 \oplus v_{12}$
  $h'_5 \leftarrow h_5 \oplus s_1 \oplus v_5 \oplus v_{13}$
  $h'_6 \leftarrow h_6 \oplus s_2 \oplus v_6 \oplus v_{14}$
  $h'_7 \leftarrow h_7 \oplus s_3 \oplus v_7 \oplus v_{15}$

Hashing the message

A given input message is padded with a bit '1' followed followed by at most 511 bits of zeros, so that the message size is equal to 447 modulo 512. This padding is followed by a bit '1' and a 64-bit unsigned big-endian representation of block length l. The padding to a message, can be represented as $m \leftarrow m \parallel 1000\ldots0001\langle l \rangle_{64}$

Algorithm 1 BLAKE Compression procedure

1: $h^0 \leftarrow IV$
2: **for** $i = 0, \ldots, N-1$ **do**
3: $\quad h^{i+1} \leftarrow compress(h^i, m^i, s, l^i)$
4: **end for**
5: **return** h^N

As shown in algorithm 3.1, the BLAKE compression function ingests the padded message block by block, in a loop starting from the initial value, and then sends the last chained value obtained from the finalization to the Ω truncation function, to obtain the hash value.

2.3 Keccak

3 Related Work

4 Methodology

5 Evaluation and expected outcomes

References

[1] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan. Blake. http://www.131002.net/blake/blake.pdf, April 2012.

[2] Eli Biham and Orr Dunkelman. A framework for iterative hash functions - haifa. Cryptology ePrint Archive, Report 2007/278, 2007. http://eprint.iacr.org/.

[3] Wikimedia Foundation. *Cryptography*. eM Publications, 2010.

[4] James Joshi. *Network Security: Know It All: Know It All*. Newnes Know It All. Elsevier Science, 2008.

[5] Douglas R. Stinson. *Cryptography Theory and Practice*, chapter 4. Cryptographic Hash Functions. Chapman & Hall/CRC, Boca Raton, FL 33487-2742, USA, third edition, 2006.

[6] Søren Steffen Thomsen, Martin Schläffer, Christian Rechberger, Florian Mendel, Krystian Matusiewicz, Lars R. Knudsen, and Praveen Gauravaram. Grøstl - a sha-3 candidate version 2.0.1. http://www.groestl.info/Groestl.pdf, March 2011.