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 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. There was little to choose amongst the 5 submissions that made it to final round.

This project, will cryptanalyse Keccak and two other finalist in SHA-3 BLAKE and Grøstl with the hill climbing algorithm. The hill climbing algorithm is generic and is hash construction agnostic. The complete version of the finalist algorithms are unbreakable by present day standards. However on the reduced version of BLAKE, hill climbing algorithm has found some success. I propose to test hill climbing on the 3 algorithms chosen, and make a claim that Keccak truly deserves to be a winner, even with a reduced version.

1 Problem Statement

2 Background

2.1 Hashing

A cryptographic hash function, is a function that can take string data of arbitrary length as input. And output a bit string of fixed length, that is ideally unique to the input string given. The aforementioned is description of a single fixed hash function. But, hash functions can be tweaked with an extra key parameter. This gives rise multiple hash functions or *hash family* as defined below. [**?**]

A *hash family* is a four-tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$, satisfying the following conditions.

- $\mathcal{X}$ is a set of possible messages
- $\mathcal{Y}$ is a finite set of hash function output
- $\mathcal{K}$, the *keyspace*, is a finite set of possible keys
- For each $K \in \mathcal{K}$, there is a hash function $h_k \in \mathcal{H}$. Each $h_k : \mathcal{X} \to \mathcal{Y}$

In the above definition, $\mathcal{X}$ could be finite or infinite set, but $\mathcal{Y}$ is always a finite set, since the length of bit string or hash function output, that defines $\mathcal{Y}$ is finite. A pair (x, y) $\in \mathcal{X} \times \mathcal{Y}$ is a *valid pair* under key K, if $h_k(x) = y$.

If $\mathcal{F}^{\mathcal{XY}}$ denotes set of all functions that map from domain $\mathcal{X}$ to co-domain $\mathcal{Y}$. And if $|\ \mathcal{X}\ |$ = N and $|\ \mathcal{Y}\ |$ = M, then $|\ \mathcal{F}^{\mathcal{XY}}\ |$ = M^N. Then any hash family $\mathcal{F} \subseteq \mathcal{F}^{\mathcal{XY}}$ is called as (N, M) - hash family.

An *unkeyed hash function* is a function $h_k : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are as defined above, and where $|\ \mathcal{K}\ |$ = 1. Thus a single fixed function h(x) = y, or an unkeyed hash function as hash family with only one key. For the purpose of this document, we will be concentrating on unkeyed hash family or fixed hash functions only, and will be referring to them as hash functions, unless mentioned otherwise.

The output of a hash function is generally called as a message digest. Since, it can viewed as a unique snapshot of the message, that cannot be replicated if the bits in message are tampered with.

2.2 Properties of an ideal hash function

An ideal hash function should be easy to evaluate in practice. However, it should satisfy the following three properties primarily, for a hash function to be considered *secure*.

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 $h(x) = y$.

The problem preimage suggests that can we find an input $x \in \mathcal{X}$, given we have the hash output y, 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 $h(x) = h(x')$.

Second preimage problem suggests that given an input x, can another input x' be found, such that $x \neq x'$ and hash output of both the inputs are same, that is $h(x) = h(x')$. 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. If the collision problem for the hash function, is computationally complex, then the hash function is said to be collision resistant.

Basically, the above properties make sure that hash function has one to one mapping from input to output, and is one way. That is if a two different input strings with even minute differences should map to two different hash values. And it should be practically infeasible, to find a input given a hash value.

Random Oracle Model

On the basis of the above properties, an ideal hash function can be abstracted as a random oracle. Random oracle model, proposed by Bellare and Rogaway, is a mathematical model of ideal hash function. It can be thought of this way, that the only way to know the hash value for an input x would be to ask the Oracle or rather compute the hash of the input itself. There is no way of formulating or guessing the hash value for input, even if you are provided with substantial number of input and output pairs. It is analogous to looking up for corresponding value of the key in a large table. To know the value for an input, you look into the table. A well designed hash function mimics the behaviour as close as possible to a random oracle.

2.3 Standards and NIST Competition

Secure Hashing Algorithm(SHA)-0 and SHA-1

SHA-0 was initially proposed by National Security Agency(NSA) as a standardised hashing algorithm in 1993. It was later standardised by National Institute of Standards and Technology(NIST). In 1995 SHA-0 was replaced by SHA-1 designed by NSA. [**?**, **?**]

In 1995 Florent Chabaud and Antoine Joux, found collisions in SHA-0 with complexity of 2^{61}. In 2004, Eli Biham and Chen found near collisions for SHA-0, about 142 out of 160 bits to be equal. Full collisions were also found, when the number of rounds for the algorithm were reduced from 80 to 62.

SHA-1 was introduced in 1995, which has block size of 512 and output bits of 160, which are similar to that of SHA-0. SHA-1 has an additional circular shift operation, that is meant to rectify the weakness in SHA-0.

In 2005 a team from Shandong University in China consisting of Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu, announced that they had found a way to find collisions on full version of SHA-1 requiring 2^{69} operations. This number was less than the number of operations required if you did a brute force search, which would be 2^{80} in this case. [**?**] An ideal hash function should require the number of operations to find a collision be equal to a brute force search, to idealize the random oracle.

Analysis was done by Jesse Walker from Skein team, on the feasibility of finding a collision in SHA-1, using HashClash developed by Marc Stevens. It was estimated that the cost for hiring computational power, as of October 2012, to find the collision, would have been $ 2.77 million. [**?**]

SHA-2

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. Where 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.

Collisions for 22-step attack on SHA-256 and SHA-512 were found with a probability of 1. Computational operations, for 23-step and 24-step for SHA-256 attack were $2^{11.5}$ and $2^{28.5}$ for the corresponding reduced version of SHA-256, have been found. For SHA-512 reduced versions the corresponding values for 23 and 24 step were $2^{16.5}$ and $2^{32.5}$. [**?**] Here steps, are analogous to rounds of compression on the input given. Since, SHA-2 family relies on the $Merkle - Damgàrd$ construction,

the whole process of creation of hash can be considered as repeated application of certain operations generally called as compression function, on the input cumulatively. The steps here refer to the number of rounds of compression applied to the input.

Preimage attacks on reduced versions of 41-step SHA-256 and 46-step SHA-512 have been found. As per the specifications, SHA-256 consisted of 64 rounds, while SHA-512 consisted of 80 rounds. [**?**] As, it can be seen, the SHA-2 functions can be said as partially susceptible to preimage attacks.

NIST competition and SHA-3

In response to advances made in cryptanalysis of SHA-2. NIST through a Federal Register Notice announced a public competition on November 2, 2007. For a new cryptographic hash algorithm, that would be SHA-3. Submission requirements stated to provide a cover sheet, algorithm specifications and supporting documentation, optimized implementations as per specifications of NIST, and intellectual property statements.

Submissions for the competition were accepted till October 31, 2008, and 51 candidates from 64 submissions for first round of competition were announced on December 9, 2008. On October 2, 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.

2.4 Applications

Applications of cryptographic hash functions, can be broadly classified in areas of verification, data integrity and pseudo random generator functions.

- **Verification and data integrity**
    1. Digital Forensics: When digital data is seized and to be used as evidence, a hash of the original digital media is taken. A copy of the digital evidence is made under the regulations, and the hash of the copied digital media is made, before it can be examined. After the evidence has been examined, then another hash value of the copy of the evidence that was used in examination is made. This ensures, that evidence has not been tampered. [**?**]
    2. Password verification: Passwords are stored as hash value, of password concatenated with some salt string. The choice of salt depends on implementation. When a password is to be verified, it is first concatenated with the respective salt. A hash value of this new modified password

string is taken and compared with the value stored in the database. If the values match, then the password is authenticated.

3. Integrity of files: Hash values can be used to check, that data files have not been modified over the time in any way. Hash value of the data file taken at a previous time is checked with the hash value of the file taken at present. If the values do not match, it means that file in question has been modified over the time period between, when hash value of the file was taken and present.

- **Pseudo random generator function:** Cryptographic hash functions can be used as pseudo random bit generators. The hash function is initialised with a random seed, and then hash function is queried iteratively to get a sequence of bits, which look random. Since, the cryptographic hash algorithm is a mathematical function, so the sequence of two pseudo random bits would be similar if they come from same hash function with the same key. And they would not be perfectly random.

3 SHA-3 finalists : Keccak, BLAKE and Groøstl

In this section we take a look at the three finalists, for the SHA-3 competition, that we will be subjecting to experimentation.

3.1 Keccak

Keccak won the SHA-3 competition and is now the new SHA-3 algorithm. Keccak is based on the sponge function. [**?**]

Sponge Functions: [**?**]

Sponge functions form the basis for Keccak, and are core to understanding its security claims. Instead of claiming security as an ideal function. They claimed for an ideal mangling function. A generic attack on the sponge function of Keccak will be considered as generic if it does not exploit the properties of the permutation. So for this section I will have to give the pseudo algorithm 1 of absorption and squeezing techniques along with the definitions. And also the diagram, that is involved. State pre-computation, can be included as a passing description. Modes are not so interesting, just the same thing said in different ways. Tree hashing, what about it? Do not have to read it, but better understand the stuff. Well the information nodes are in leaves, and F applied to them, and C bits taken off them. The internal node keeps track of the sons and then concatenates the C bits, and applies F. Process recursively done, to get the last value. How do you handle input

which is large and with fixed number of nodes, accept growing number of input blocks. Now we get to the part of duplex application, which we talked about before, but why the fuck I am doing it?

3.2 BLAKE

3.3 Grøstl

4 Related work

4.1 Zero Sum Distinguishers

Zero sum distinguishers were first presented in CHES 2009 rump session [**?**]. A zero sum distinguisher, for any function is a way to find a set of values, that sum to zero, such that their respective images also sum to zero.

4.2 Cryptanalysis done on Keccak

4.3 Cryptanalysis done on BLAKE

4.4 Cryptanalysis done on Grøstl

5 Hypothesis

For the time being, here is my hypothesis, or the premise of my question. Keccak has been selected over BLAKE and Groestl for what? Is there is a basis that Keccak's property is still comparitively immune to zero sum distinguisher compared to BLAKE and Groestl in the reduced versions. This is what, I would like to find out and examine.

6 Research Approach and Methodology

6.1 Architecture

Not sure how this will be, but I will have to pull my socks up and see, let us say that there will be one class and that will see over the implementation of all the other implementation of the algorithm. Then how do you do the zero sum distinguisher thing. Well so you will have to input and output and see what you want, from them.

6.2 Platform, Languages and Tools

Well for starters, my platform will be Ubuntu Linux and will try to make it work on the CS systems at RIT. So the C++ or Go-lang will be selected for implemenation and experimentation based on what I can do or not.

6.3 Proposed schedule

Well once I give away my proposal. Then the clock starts. Week 1, try to get the things going. Implementation of all the three algorithms if possible in 1 and 1/2 week. Next, 1 week look at the zero sum distinguisher implementation and how to get the things done. Next 1 week do the input and output and at the same time keep writing the report in parallel.

7 Evaluation and expected outcomes

So I will be running experimenting with the same inputs, on all three of the algorithms and then will try and check for how much of preimage can I figure out for them. So there will be comparative graphs of all three. But this does not give a complete picture given, that they have different rounds and structure, so you really cannot make a comparison given that different algorithms reduced to different strengths. So this I will have to clear with my advisor.