REST Journal on Emerging trends in Modelling and Manufacturing
Vol: 5(3), 2019
REST Publisher
ISSN: 2455-4537

Website: www.restpublisher.com/journals/jemm

An Approach towards Design and Implementation of Cryptographic Hash Techniques for Message Authentication

Soumya Bishnu , Rajdeep Chakraborty
Dept. of Computer Science Engineering, Netaji Subhash Engineering College, Kolkata, WestBengal,India
rajdeep_chak@rediffmail.com

Abstract

This method seeks to improve the speed of the hash function particularly when a large set of messages with similar blocks such as documents with common headers are to be hashed. The method utilizes the peculiar run time reconfigurability feature of FPGA. Essentially, when a block of message that is commonly hashed is identified, the hash value is stored in memory so that in subsequent occurrences of the message block, the hash value does not require to be recomputed; rather it's simply retrieved from memory, thus giving significance increase in speed.

Key Words: Cryptographic hash function, Message Authentication, Fast Hash, SHA1, MD5, Hash Throughput

1. Introduction (Cryptographic Hash Function)

A cryptographic hash function [1] is a special class of hash function that has certain properties data of arbitrary size to a bit string of a fixed size (a hash) and is designed to be a one-way which make it suitable for use in cryptography. It is a mathematical algorithm that maps function, that is, a function which is infeasible to invert. The only way to recreate the inputdata from an ideal cryptographic hash function's output is to attempt a brute-force search of possible inputs to see if they produce a match, or use a rainbow table of matched hashes. Bruce Schneier has called one-way hash functions "the workhorses of moderncryptography".[1] The input data is often called the message, and the output (the hashvalue or hash) is often called the message digest or simply the digest. Hash function like SHA1 and MD5, which are widely accepted are inefficient in lightweight and IoT security environment due to huge gate requirements. The main objective of the proposed algorithms are to make it less complex in terms of GEs and yet comparable to SHA1 and MD5. In this paper three cryptographic hash functions are proposed with comparable result for SHA1 and MD5. Thus three algorithms are proposed in this paper namely, Proposed Algorithm 1, Proposed Algorithm 2 and Proposed Algorithm 3. Section II describes the three classifications of hash, Section III briefly describes Objective and Goal of this work, Section IV come up with implementation as project functionality. Section V gives the results of the proposed algorithms, Section VI gives the conclusion and Future work and finally references are added in the end.

2. Classification of Cryptographic Systems

A. Unconditional security (perfect secrecy) Unconditional security (perfect secrecy)

This category is based on the information theory published by Shannon back in the 40s.To have perfect secrecy on a cryptographic system [2,3] means that if the attacker has the plaintext and the cypher text, it's of no use for him to cryptanalyze it. It means that these values are independent random variables. To achieve this, we need a third element which will be the secret information known by the sender (the key to crypt) and the receiver (the key to decrypt). The problem, stated also in Shannon's publication, is the impracticability of this solution, because it needs the key to be as big as the message. Also, the key cannot be used to exchange different messages, meaning that we should use different keys every time. The classic example of perfect secrecy is One-Time Pad, invented in 1917 by J. Mauborgne and G. Vernam.

B. Provable Security

Cryptographic systems are said to be provably secure [4] if the fact to break it is as difficult as to break a generic basic hard-problem (i.e. factoring big numbers, calculate square roots in modulo a composite, calculation of discrete logarithms on a finite group, etc). The most famous cryptographic system in this category is Rabin's cryptosystem. Notice that RSA is not proven to be provable secure, just in the random oracle model [5].

C. Computational Security

Most of the cryptographic systems [2] are in this category. Computational security means that the effort needed to break it is not available to possible attackers; or that the potential attackers don't have the sufficient amount of resources to break it.

3. Objective and Goal

The basic concepts of cryptography are treated quite differently by various authors, some being more technical than others. Brassard [Brassard G 1988] provides a concise, lucid, and technically accurate account. Schneier [Schneier B 1996] gives a less technical but very accessible introduction. Salomaa [Salomaa A 1990], Stinson [Stinson D 2006], and Rivest [Rivest R L 1990] present more mathematical approaches [8],[9]. Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication and data origin authentication. So the main goals of cryptography are privacy or confidentiality, data integrity, authentication and non-repudiation. Privacy or confidentiality is the service used to keep the content of information secret from all but those authorized one to have it. Secrecy, confidentiality and privacy are synonymous terms. There are number of approaches to providing confidentiality through mathematical algorithms which render data unintelligible [Santhosh Kumar 2010].Data integrity refers to the unauthorized manipulation of data. Data manipulation includes such things as insertion, deletion and substitution. It ensures the ability of detecting data manipulation by unauthorized parties. Authentication is a service related to identification. This function [3], [4] applies to both entity authentication and data origin authentication. Two parties entering into a communication should identify each other. Moreover, information delivered over a channel should be authenticated as to origin of data, data content, time sent etc. Non-repudiation is a service which prevents an entity from denying previous commitments or action. When disputes arise due to an entity denying that certain actions are to be taken, a means to resolve the situation is necessary. The term information security is much broader, encompassing such things like authentication and data integrity. The basic terms of information security are: An information security service is a method to provide some specific aspects of security. For example, integrity of transmitted date is a security objective, and a method to ensure this aspect is an information security service. Breaking an information security service (which often involves more than simply encryption) implies defeating the objective of the intended service. A passive adversary is an adversary who is capable only of reading information from an unsecured channel. An active adversary is an adversary who may also transit, alert, or delete information on an unsecured channel. An encryption [6] scheme is said to be breakable if a third party, without prior knowledge of the key, can systematically recover plaintext from corresponding cipher text within some appropriate time frame. An appropriate time frame will be a function of the useful life span of the data being protected [Schneier B 1996] The main goal of this project is to study different algorithm and how it works. Post this a new algorithm will be designed which is expected to work similar. Also its time complexity will be validated with respect to other existing algorithms.

4. Project Functionality

We are in process of designing new algorithms that will take a file or a string format data type as an input and would generate a hash code[5][7]. This hash code will be unique to that particular set of input. Also the time required to calculate the code to generate the hash value is tracked. Now it will be compared to SHA-1 and MD5 to analysis how good it is compared to the others.

A. One-way Functions

One-way functions are important. A function is a one-way function if:

Vx □ X, f(x) = y

is easy computed, but it is virtually impossible to find a function g where g(y) = x

The only problem in this definition is the sentence 'is virtually impossible'; it is important to define the impossibility of a solution. There is a whole theory behind one-way functions based on hard problems. One-way functions are most used in asymmetric cryptography. Difficulty to find the inverse of the function depends on the size of the key used. So, that's why we feel safer using a 2048bit key than a 512bit key, because we are actually in safer ground, of course, in terms of confidentiality of the crypted messages.

Some examples of one-way functions are:

• Calculation of square roots modulo a composite:

f(x) = x2mod n where n = pq with n, q unknown

• A one-way function can be easily built based on a block cryptography system E (like DES):

y = f(x) = Ek(x) ⊕ x where k is fixed and known; E is the function that crypts.

B. Hard Problems

Use either SI (MKS) or CGS as primary units. (SI units are Complexity theory has long studied problems like: How do we know if a program will stop and give an answer? At first, this seems difficult, but it turns that it is not only a difficult, but an impossible problem. That's why we still don't have a program that tells us if a program will hang or not in a reasonable time.

Mathematicians have then classified the hard problems. There are the problems that can be solved in polynomial time (or sub-exponential), it means, that the time it would take to solve them is known, and this time grows in a polynomial way as the problem becomes more complicated (these are called P problems). And there are the other problems, called NP that are problems [10] whose solutions can be checked in polynomial time.

The question of the millennium (for which you could win a nice prize) is to know whether

P = NP or not.

There is still another type of problems, called NP-complete. When we can reduce any problem B NP to a problem A NP then we can say A is NP-complete. Some NP-complete problems are the Traveling Salesman problem and the Knapsack problem. So, basically, in order to solve the big question P = NP? We should find a NP complete problem that can be solved in polynomial time. Then we would be able to reduce all other NP problems to our solution and solve them. A lot of work has been made to find an answer with little results. If sometime P = NP should be proven, we would be able to calculate very difficult problems in polynomial time without having to develop the computing technology.

C. Proposed Work

The proposed work for this template consists of wrap over algorithms. We have assumed a 256 bit hash function, with 512 bits of internal state. This means n = 256. Simple arithmetic shows that if we choose nine rounds we have 5.59/512 = 8995 as a safety factor. Nine rounds is about twice as many as what would be optimal, but our nonlinear mix function isn't perfect. If the number of bits is increased, it turns out that nine rounds is still good up until unreasonably large hash sizes. For example, let n = 8192 (an eight kilobyte hash), then we have 10.59/16384 = 94686; where it looks like we could decrease by a round. At even higher (completely unrealistic) values more rounds are needed though. We have instantiated a C++ template class like:

double_uint<__uint128_t, unsigned long long, 256> u256; to define our unsigned 256 bit type from the built-in unsigned 128 bit types that gcc has. There are 3 approaches proposed for constructing the mix function for problem solving:

i. Approach :1.

A calculation is performed on the fold applied on the offset

```
o1 += o2.fold(t1);
o2 += o1.fold(t2);
o1 += o2.fold(t3);
o2 += o1.fold(t4);
o1 += o2.fold(t5);
o2 += o1.fold(t6);
o1 += o2.fold(t7);
o2 += o1.fold(t8);
o1 += o2.fold(t9);
```

After this is set up the clock time is to be set up. Then the calculation will be done based on 32 bits of the input offset.

```
hash_step(p1[i] ^ t2, t1 ^ p2[i], t1, t2);
```

In this approach if the inputs are both zero, each step involves a folded multiply by a constant with zero. This causes us to add zero to the other half each time. (We could choose xor, but addition mixes bits slightly more.)

ii. Approach :2.

A XOR calculation is now applied over the rounds, So when zero will occur we will get an alert. Moreover there will be no exact Fixedpoints.

```
o1 += o2.fold(t1) ^ t1;
o2 += o1.fold(t2) ^ t2;
o1 += o2.fold(t3) ^ t3;
o2 += o1.fold(t4) ^ t4;
o1 += o2.fold(t5) ^ t5;
o2 += o1.fold(t6) ^ t6;
o1 += o2.fold(t7) ^ t7;
o2 += o1.fold(t8) ^ t8;
o1 += o2.fold(t9) ^ t9;
```

Here, the problem with zero is fixed, but now if both halves are equal to one, then we have a similar issue.

$$1 \times t = t.\ t \wedge t = 0$$

This has been resolved on the next approach.

iii. Approach :3.

Here we placed the XOR operation inside the folded multiply. So it now looks like:

```
o1 += (o2^t1).fold(t1);
o2 += (o1^t2).fold(t2);
o1 += (o2^t3).fold(t3);
o2 += (o1^t4).fold(t4);
o1 += (o2^t5).fold(t5);
o2 += (o1^t6).fold(t6);
o1 += (o2^t7).fold(t7);
```

o2 += (o1^t8).fold(t8);
o1 += (o2^t9).fold(t9);

Now that we have overcome all the issues we can use this to frame the result set. We will scale the numbers so that the top bit is always a one. This maximizes the size of the resulting product, giving more overlap in the fold operation. Similarly, we will forcibly set the least significant bit to always be a one. The combination of the above will yield a full 2n-bit product, rather than having padding zero-bits on the top or bottom.

5. Result and Analysis

The result and analysis is broadly given in two sections, namely, Section A illustrates the hash values received and various hash throughput after executing SHA1, MD5 and three proposed algorithms and Section B illustrates the graphical representation of First Trial and Second Trial.

A. Hash Value Generation

After the algorithm design is done the result is compared to the existing algorithms like MD5 and SHA1.

MD5 Algorithm First Trial; File size = 11539 bytes
HashValue = 98B62AB5CD7D3BBF13D529FE5BDAC629
Average Running Time = 3.3644 millisecond
MD5 Algorithm Second Trial; File size = 11539 bytes
Hash Value = 577CA8666C87EAE17E6A053FE3812E4C
Average Running Time = 3.674242857143 millisecond
SHA1 Algorithm First Trial; File size = 11539 bytes
Hash Value = 32E527EBBFD81F6B7739A87F88A4E4AB208DBD50D170DE0D0DF772A60C79529E
Average Running Time = 5.198342857143 millisecond
SHA1 Algorithm Second Trial; File size = 22528 bytes
Hash Value = 25B0F833564169D56741059DCCFA09ACCF891DCFC8042323090D83BD08D29FA4
Average Running Time = 6.25057142857 millisecond
Proposed Algorithm 1 First Trial; File size = 11539 bytes
Hash Value : 822d36d7645286032608ac6ca42c0fd8
Average Running Time = 8.698452834132 millisecond
Proposed Algorithm 1 Second Trial; File size = 22528 bytes
HashValue= 5e8619174b804a2732925f18210e769cb76a48cc
Average Running Time = 8.23037128456 millisecond
Proposed Algorithm 2 First Trial; File size = 11539 bytes
HashValue= 822d36d7645286032608ac6ca42c0fd84021557b
Average Running Time = 7.998342673443 millisecond
Proposed Algorithm 2 Second Trial; File size = 22528 bytes
HashValue = 6f0b09f81013c71cb4ccef51bb643c5cbca3dc64
Average Running Time = 8.895059842889 millisecond
Proposed Algorithm 3 First Trial; File size = 11539 bytes
HashValue= 822d36d7645286032608ac6ca42c0fd84021557b6a5ef232
Average Running Time = 9.198742858453 millisecond
Proposed Algorithm 3Second Trial; File size = 22528 bytes
HashValue= 43a21abf593e939a366a2edcff30a080bbbccfa903680518e4b90c9de68cde9f
Average Running Time = 10.25057142857 millisecond

B. Graphical Analysis

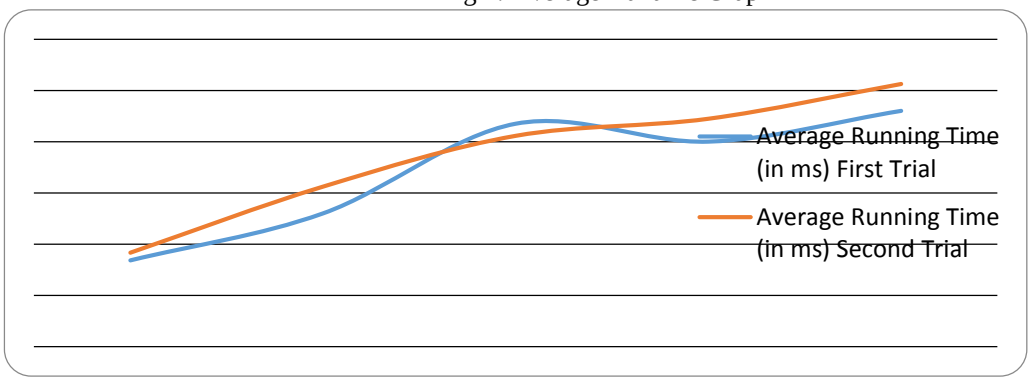
When we plot the average running time of the algorithms on the graph we can see a much broader picture.

Table I gives the average running time and it can be inferred that the result of proposed algorithms are comparable with that of standard algorithms. Fig 1 is the pictorial representation of average running time and it can be inferred that the average running time of the second trial gives the most optimal result than that of the average running time of the first trial.

Table I: Average Running Time

Algorithm	Average Running Time (in ms)	
	First Trial	Second Trial
MD5	3.3644	3.6742
SHA1	5.1983	6.2505
Proposed Algorithm 1	8.6984	8.2303
Proposed Algorithm 2	7.9983	8.8950
Proposed Algorithm 3	9.1987	10.2505

Fig 1: Average Runtime Graph



6. **Conclusion**

These proposed algorithm are tested with C and the verified with SHA-1 and MD5. Thus a comparable result is obtained and future scope is to implement it in VHDL and Python for embedded and IoT systems.

References

[1] B. V. Rompay, "Analysis and Design of Cryptographic Hash functions, MAC algorithms and Block Ciphers", Ph.D. thesis, Electrical Engineering Department, KatholiekeUniversiteit, Leuven, Belgium, 2004. (PhD Thesis)

[2]Rajdeep Chakraborty, JK Mandal, "FPGA Based Cipher Design & Implementation of Recursive Oriented Block Arithmetic and Substitution Technique (ROBAST)" (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 2, No. 4, 2011, pp 54 – 59

[3] William Stallings, "Cryptography and Network Security", 4th Ed., 2005 (Book)

[4] Yanduo Ren, Jiangbo Qian, Yihong Dong, Yu Xin and Huahui Chen, "AVBH: Assymetric Learning to Hash with Variable Bit Encoding", Published in Security and Communication Network, Hindawi, Scientific Programming, Volume 2020, Article ID 2424381, pp 1-11 pages, (SCIE and Scopus)

[5] Shiguang Liu, Ziqing Huang, "Efficient Image Hashing With Geometric Invariant Vector Distance for Copy Detection", ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), December 2019 Article No.: 106 https://doi.org/10.1145/3355394 (SCIE, SCIMago and Scopus)