Cryptographic Primitives and Secure Hash Functions

Zabeer Rahman

2021

1 Introduction

Cryptography concerns mathematical applications to ensure information security and secure communication in the presence of malicious adversaries. Cryptography can be used to hide data, making confidential information valuable and sought after. Encryption is and has been used widely to conceal plain-text messages and information all around the world. Face-to-face interactions have the freedom of exploring physical privacy, while secrets that are not sent or communicated are best kept, in this way cryptography is the base of digital privacy while still being able to send valuable information long distances.

Cryptography revolves around algorithms which are designed based on the difficulty of the mathematical operation that is involved. Adversaries are assumed to be equipped with the knowledge and technology required to break said algorithms. It may be possible in theory to break these cryptographic algorithms, but the time, power, and memory which is required to do so is so unrealistic that computer scientists have agreed to call said algorithms ”well-designed systems”. ”Well-designed systems” refer to cryptographic algorithms that are proven infeasible to break/solve even with powerful computer processors attempting to brute-force and even using inverse functions to try and solve them mathematically.[2][3]

Most currently popular algorithms have security that relies on one of three very difficulty math problems:

- Discrete Logarithms
- Integer Factorization
- Elliptic-Curve Discrete Logarithms

The only way to solve these functions easily is with quantum computers, and none with known computing power exist *yet*. Only two types of Cryptographic Primitives (to be discussed) are considered relatively secure against attacks from quantum computers *as of now*: Symmetric Key algorithms and Hash functions (both to be discussed).[2][4][6]

This paper will discuss every type of cryptographic primitive and then explore the mathematics of Secure Hash Algorithms. Commonly referred to as **SHA**, this family of algorithms is published and widely accepted as a secure standard of encryption. Hash functions are just one type of a Cryptographic Primitive, all of which must be defined before exploring the mathematical theory behind SHA.

2 Different Types of Cryptographic Primitives

A Cryptographic Primitive is defined as a low-level algorithm that is used to build cryptographic protocols and computer security systems. They are designed to do one singular task and are used as building blocks to create a secure environment. From authentication, to non-repudiation, to private key management, any protocol or system that needs security use primitive algorithms defined as separate secure steps and summing up to a safe system overall. Primitives are assumed to be reliable (using encryption/decryption routines to test how many iterations of computer operations it takes for the primitive to fail) since they are a category of algorithms that are acceptable to use for security standards. Since primitives only perform one task and are meant to perform it extremely securely, researchers can isolate the primitives from entire cryptographic systems to analyze their security and how easy it is to break them. Only when combined with security systems can primitives perform several tasks together.[7]

2.1 Digital Signatures

Digital Signatures are a standard element of many security protocols, and probably the most widely used within computer systems designed for transaction of information/assets. A digital signature (a login, a pass code, a PIN, or even a secret button combination) allows for verification of identity and authentication of accessibility permissions. A layer of validation and security is necessary specially when information/assets are being sent through a non-secure channel.[7][9]
The math behind digital key signatures concerns key generation, a signing algorithm, and a signature verifying algorithm.

2.2 Symmetric Key

Symmetric Key algorithms are a standard of encryption where the same key can be used to encrypt and decrypt a message. The keys may be the same or they may be inverses or natural transformations of each other for reversal. Symmetric key algorithms are susceptible to many computerized attacks as well as physical attacks such as eavesdropping. Specially designed keys and functions can greatly increase the security and risk of being broken if the method of attack is known prior.[2][6][7]
The math behind symmetric Key encryption makes use of stream and block ciphers to map every element of a message to elements in the cipher-text.

2.3 Asymmetric Key

Asymmetric Key algorithms are a standard of encryption where a pair of keys are used to both encrypt and decrypt a message. Usually there is a public key (known by everyone) and a private key (known by owner/sender of message). Any one can encrypt a message using a person's public key, but only the person with the private key can solve the inverse algorithm and produce the message in plain-text using cipher-text.[7]
The math behind asymmetric key algorithms makes use of prime numbers, logarithms/exponentiation, and modular arithmetic to produce one-way functions which are extremely hard to brute-force and become harder to break as numbers and values and message length grow.

2.4 One-way Hash Functions

One-way Hash Functions are algorithms which takes a message of arbitrary size and maps it to a fixed size bit array. Defined cryptographic hash functions are deterministic (same input produces same output every time), utilize the avalanche effect (small change in input

implies big change in output), efficient and quick, irreversible, and collision resistant (no two inputs produce the same output). Many famous and verified hash algorithms exist such as *BLAKE2b, MD5, SHA-3, RIPEMD-320, GOST, Whirlpool, RadioGatún, etc.* The difference between them in the sizes of the outputs, internal states, blocks, words, lengths, and also the number of hashing rounds that occurs in the algorithm. The only way to break a one-way hash function is to brute-force search and use trial-and-error to see if possible inputs produce a matching hash output.[5][6][7]
The math behind one-way hash functions will be further explored in Section 3.

2.5 Private Information Retrieval

Private Information Retrieval is a standard of communications when a user interacts with a server where the user is retrieving a data item and the server in possession of the database is not informed which data item is retrieved. The only way to ensure entire privacy would be to send an entire copy of the database to the user, but that would compromise the security of all the other database items, so users must request blocks of information which are relevant. Many different systems exist, with different orders of operations from requests/approvals to actually sending the information and sifting through all the irrelevant data for the useful data. Private Information Retrieval intersects with Byzantine attacks where adversaries take malicious control of a number of authenticated devices that are part of a distributed computing system.[7]
The math behind Private Information Retrieval lies in different protocols defined differently for each party, but by using finite series and vector math researchers can define a non-colliding system which seems to be the best use-case for this primitive.

2.6 Commitment Scheme

Commitment Scheme is a cryptographic primitive which locks in a user's committed value so that it can not be appended or changed alter. The value can sometimes be revealed in the future, but they are binding, making them immutable.[7]
The math behind Commitment Schemes depends on the scenario. Sometimes it is as simple as probability and a permutation to find out the chances that the committed value is a certain number, but other times the math is almost non-existent at first but becomes important such as with zero-knowledge proofs which depend on information the user learns throughout. Still yet, there are way to use logarithms/exponentiation, statistics, and bitwise operations to ensure security of hidden values no matter the environment or strength of adversary.

2.7 Pseudo-random Number Generator

Cryptographically Pseudo-Random Number Generator is a method for creating a "random" sequence of numbers or symbols for the use in a cryptographic system. Examples could be a key generator, a nonce (a one-time use number used to identify an instance of a certain action), or even temporary digital signatures.[7]
The math behind Pseudo-random Number Generators is broken down into 2 categories. There exist truly randomly generated numbers which take in environmental factors and

measure them with sensors to parse the data into numerals which map to combinations and permutations of symbols. Then there are *pseudo*randomly generated numbers which produce apparent random numbers/symbols but they are determined by an initial value called the seed value. The seed value determines which block is chosen from the string of long sequence of numbers and symbols in every permutation, giving that segment a "random" feel. However, if the same seed value is used in a pseudo-random number generator twice it should produce the same randomly generated number.

2.8 Mix Network

Mix Networks are a routing standard which creates communications using proxy servers communicating on behalf of the client, and mixing the messages received from different senders and sending them to their destination in a random order, making the digital trail extremely hard to trace.[7]
The math behind Mix Networks concerns sealing messages (encrypting) using a public key and then appending the destination address with the cipher-text, and then encrypting the concatenation again using the public key. This creates a nested encryption only accessible by the proxy server's private key which lets it unseal and decrypt both the destination and the message to send.

3 Secure Hash Algorithm 1

SHA stands for Secure Hash Algorithm and refers to the family of one-way hash functions that are published and used within computer science and mathematics. They inherit every characteristic of one-way hash functions as previously defined. This section focuses on SHA-1 and how a message is mathematically transformed into a hash.

3.1 How it Works

SHA-1 takes a message of arbitrary length and produces a 160-bit hash value that is interpreted as a hexadecimal number with 40 characters.
For the purpose of efficiency and this demonstration, string "A Test" will be manually converted into a hash using SHA-1.

1. Take input text and split into array, then convert to ASCII character codes
   (a) "A Test"
   (b) [A, , T, e, s, t]
   (c) [65, 32, 84, 101, 115, 116]
2. Convert ASCII codes to binary, then pad zeros to front until strings are 8 bits long
   (a) [1000001, 100000, 1010100, 1100101, 1110011, 1110100]
   (b) [01000001, 00100000, 01010100, 01100101, 01110011, 01110100]

3. Concatenate strings and append a 1
   (a) 0100000100100000010101000110010101110011011101001
4. Append binary string and pad zeros to end until value is congruent to 512(mod 448)
   (a) 01000001001000000101010001100101011100110111010010000000000000
   0000000000000000000000000000000000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000
   0000000000000000000
5. Take 8-bit ASCII code array from step 2.b and compute length, then convert length to binary
   (a) [01000001, 00100000, 01010100, 01100101, 01110011, 01110100], length = 48
   (b) length *in binary* = 110000
6. Append binary length and pad zeros to front until 64 characters long
   (a) 0000000000000000000000000000000000000000000000000000000000110000
7. Append binary length to previously created binary string (message)
   (a) 01000001001000000101010001100101011100110111010010000000000000
   0000000000000000000000000000000000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000
   0000000000000000000 00000000000000000000000000000000000000000
   0000000000000000000000000000000110000
8. Break string into blocks of 512 characters, then break blocks into chunks of sixteen 32-bit objects
   (a) 01000001001000000101010001100101 01110011011101001000000000000000
   00000000000000000000000000000000 00000000000000000000000000000000
   00000000000000000000000000000000 00000000000000000000000000000000
   00000000000000000000000000000000 00000000000000000000000000000000
   00000000000000000000000000000000 00000000000000000000000000000000
   00000000000000000000000000000000 00000000000000000000000000000000
   00000000000000000000000000000000 00000000000000000000000000000000
   00000000000000000000000000000000 00000000000000000000000000110000

9. Use XOR bitwise operation on blocks of 16 words continuously and append each new word (xor) to extend string until length is 80 32-bit objects (words)
    (a) xor 1 = word 1 XOR word 2
    (b) xor 2 = xor 1 XOR word 3
    (c) xor 3 = xor 2 XOR word 4
    (d) *and so on...*

10. Declare and initialize variables for use in SHA-1 (always pre-defined)
    (a) h0 = 01100111010001010010001100000001
    h1 = 11101111110011011010101110001001
    h2 = 10011000101110101101110011111110
    h3 = 00010000001100100101010001110110
    h4 = 11000011110100101110000111110000

11. Loop through using XOR bitwise operators on SHA-1 h variables and blocks of words, then reassign SHA-1 h variables those new values
    (a) ((h0 XOR word 1) XOR word 2)... XOR word 80) = 10001111000011000000100001010101
    (b) h0 = 10001111000011000000100001010101
    h1 = 10010001010101100011001111100100
    h2 = 10100111110111100001100101000110
    h3 = 10001011001110000111010011001000
    h4 = 10010000000111011111000001000011

12. Convert each resulting SHA-1 variable to hexadecimal and concatenate the strings
    (a) h0 = 8f0c0855 h1 = 915633e4 h2 = a7de1946 h3 = 8b3874c8 h4 = 901df043
    (b) **HASH VALUE:** 8f0c0855915633e4a7de19468b3874c8901df043

4 Implications

Several different steps involved in the process of SHA-1 make it very evident as to why one-way hash functions have the characteristics they have. Step 4 makes it clear how the output is data of a fixed length because it is ensured that the string's value is congruent to 512, so data can be padded and split up no matter how big or small the incoming data is. Furthermore, step 11 reinforces the fixed length output by keeping variable reassignment to the original five pre-defined variables[5].

SHA-1 was widely used through the mid-2000's but is now not considered secure against ”well-funded” opponents. Computers strong enough have found collisions with complexity of 2 raised to the 60th power. The brute-force collision search is still the only known way to break a Secure Hash Algorithm intuitively, but SHA-1 is vulnerable to chose-prefix attacks.

It was chosen to simply illustrate how a one-way hashing function works in terms of math. The publishers of SHA-1 have also release SHA-2 and SHA-3 after several years of research and collaboration by mathematicians and computer scientists internationally. SHA-2 is a family of Secure Hash Algorithms which use fixed output bit lengths that are equivalent to powers of 2.SHA-3 is the latest member of the algorithmic family, and utilizes more rounds of reassignment, and more variables and words and blocks as defined in the SHA-1 process. SHA-3 has the best security against both collision attacks and length extension attacks (attacks on messages which do not need to know the content of the message). Every string and value worked with in Section 3.1 would be much much larger, and impossible to do manually. Because of this, SHA-3 is replacing SHA-1 wherever possible unless the hashing algorithm is not being used for security reasons but rather data storage/execution efficiency. SHA-2 and SHA-3 can be commonly seen in blockchain and network security applications because of their ability to form hashing merkle trees where data chunks and their accessibility depend on the validity and and authenticity of previous data chunks.[5][8][10]

References

[1] “#1 Solidity Tutorial Ethereum Blockchain Programming Course.” *CryptoZombies*, cryptozombies.io/.

[2] Blahut, Richard E. *Cryptography and Secure Communication*. Cambridge Univ. Press, 2014.

[3] Buttyán, Levente, and István Vajda. *Kriptográfia És Alkalmazásai (Cryptography and Its Applications)*. Typotex, 2005.

[4] “Crypto101.” *Crypto 101*, www.crypto101.io/.

[5] “Detailed SHA-256 Algorithm Explanation.” *YouTube*, YouTube, 27 Apr. 2020, www.youtube.com/watch?v=PMOEdd4yzyU.

[6] Dobbertin, Hans, et al. “RIPEMD-160: A Strengthened Version of RIPEMD.” *Fast Software Encryption*, 1996, pp. 71–82., doi:10.1007/3-540-60865-6_44.

[7] Hajny, Jan, et al. “Performance Evaluation of Primitives for Privacy-Enhancing Cryptography on Current Smart-Cards and Smart-Phones.” *Data Privacy Management and Autonomous Spontaneous Security*, 2014, pp. 17–33., doi:10.1007/978-3-642-54568-9_2.

[8] Ma, Jun, et al. “SHAvisual.” *Proceedings of the 2014 Conference on Innovation Technology in Computer Science Education - ITiCSE '14*, 2014, doi:10.1145/2591708.2602663.

[9] Menezes, Alfred J., et al. *Handbook of Applied Cryptology*. CRC, 1997.

[10] Morawiecki, Paweł, et al. “Rotational Cryptanalysis of Round-Reduced Keccak.” *Fast Software Encryption*, 2014, pp. 241–262., doi:10.1007/978-3-662-43933-3_13.