Exploring Cryptographic Primitives and Secure Hash Functions: A Comprehensive Overview

Riddhiman Bhattacharya

September 7, 2022

1 Introduction

Cryptography is a mathematical approach to keeping information safe and secure when there are people who might try to harm it. It's like a secret code that can hide important data, making it valuable and desirable. Encryption is a common method used to hide messages and information, ensuring that only the intended recipients can understand them. In face-to-face conversations, we have the advantage of physical privacy, but when we need to send information over long distances, cryptography is essential for keeping it private and secure.

Cryptography is all about using special mathematical operations to create algorithms. These algorithms are designed to be really hard to crack. When we talk about adversaries, we assume they have the knowledge and technology to try and break these algorithms. In theory, it's possible to break them, but it would take an unrealistically long time and require so much computing power and memory that it's just not practical. That's why computer scientists consider these algorithms as "well-designed systems." These systems have been proven to be incredibly difficult to break, even if you use super-powerful computers and try to solve them using math tricks.[2][3]

Most currently popular algorithms have security that relies on one of three very difficult 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 cover various types of cryptographic techniques and then delve into the mathematics behind Secure Hash Algorithms, commonly known as **SHA**. SHA is a group of algorithms that are widely accepted and considered secure for encryption. Hash functions are a specific type of cryptographic technique, and it is important to define all the different types before delving into the mathematical theory behind SHA.

2 Different Types of Cryptographic Primitives

A Cryptographic Primitive is a basic algorithm used to create secure computer systems and protocols. These algorithms are designed to perform a single task and are used as building blocks to ensure security. They are used for tasks like authentication, non-repudiation, and managing private keys. Primitives are considered reliable and secure, and their strength is

tested by measuring how many computer operations it takes for them to fail. By isolating primitives from larger cryptographic systems, researchers can analyze their security and vulnerability. Primitives work together with security systems to perform multiple tasks and create a secure environment.[7]

2.1 Digital Signatures

Digital signatures are an essential part of many security protocols and commonly used in computer systems for exchanging information or assets. They provide a way to verify identity and authenticate permissions, similar to a login, passcode, PIN, or secret button combination. In situations where information or assets are transmitted through an insecure channel, a digital signature adds an extra layer of validation and security.[7][9]
.

The mathematics behind digital key signatures involves three main components: **key generation**, a **signing algorithm**, and a **signature verification algorithm**.

1. **Key generation**: Process of creating a pair of cryptographic keys:

   - **Private Key**: The private key is kept secret and used to generate digital signatures

   Corresponding Public Key: The public key is freely shared and used to verify those signatures.

2. **Signing algorithm**: Takes the private key and the data to be signed as input and produces a digital signature. This algorithm applies mathematical operations that ensure the uniqueness and integrity of the signature.

3. **Signature verification algorithm**: Takes the public key, the data, and the digital signature as input and checks if the signature is valid. It uses mathematical computations to verify the integrity of the signature and ensure that it was generated using the corresponding private key.

2.2 Symmetric Key

Symmetric Key algorithms are encryption standards that use the same key for both encrypting and decrypting a message. The keys can be identical or related in a reversible way. These algorithms are vulnerable to various computer-based attacks and physical eavesdropping. By using specially designed keys and functions, the security can be significantly enhanced, but the risk of being broken increases if the method of attack is known in advance.[2][6][7] In symmetric key encryption, the mathematical principles involve stream and block ciphers to transform each element of a message into corresponding elements in the cipher-text.

2.3 Asymmetric Key

Asymmetric Key algorithms are encryption standards that involve a pair of keys for both encrypting and decrypting a message. In this method, there is a public key, known by everyone, and a private key, known only by the owner or sender of the message. Any

person can encrypt a message using the recipient's public key, but only the person with the corresponding private key can decrypt the message and obtain the original plain text from the encrypted cipher-text.[7]
Asymmetric key algorithms rely on prime numbers, logarithms/exponentiation, and modular arithmetic to create one-way functions. These functions are very difficult to crack through brute-force attacks, and their resistance increases as the numbers, values, and message length involved grow.

2.4 One-way Hash Functions

One-way Hash Functions are algorithms that transform a message of any size into a fixed-size bit array. These cryptographic hash functions possess several key characteristics: determinism (the same input always yields the same output), the avalanche effect (a small change in input results in a significant change in output), efficiency and speed, irreversibility, and collision resistance (no two inputs produce the same output). There are numerous well-known and validated hash algorithms, including **BLAKE2b, MD5, SHA-3, RIPEMD-320, GOST, Whirlpool, RadioGatún**, among others. The variations among these algorithms lie in the output sizes, internal states, blocks, words, lengths, and the number of hashing rounds employed in the algorithm. The only way to break a one-way hash function is through brute-force search and trial-and-error to determine if possible inputs yield 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 (PIR) is a communication standard that enables users to retrieve specific data items from a server without disclosing which item is being retrieved. The challenge lies in maintaining complete privacy without compromising the security of the entire database, which would occur if the server sent a full copy of the database to the user. Instead, users request relevant blocks of information to minimize exposure.

There are various PIR systems with different sequences of operations, ranging from request and approval to the actual transmission of information and filtering out irrelevant data to extract the desired information. PIR also intersects with Byzantine attacks, where adversaries gain malicious control over authenticated devices within a distributed computing system.[7]
Private Information Retrieval protocols utilize various mathematical techniques, and researchers often employ finite series and vector math to establish non-colliding systems, which prove to be advantageous for this primitive.

By leveraging finite series and vector math, researchers can design protocols that allow users to retrieve desired information from a server without revealing the specific item being retrieved. These mathematical techniques help ensure that the system minimizes collisions, meaning that multiple users accessing the server simultaneously do not interfere with each other's privacy.

These non-colliding systems, built upon mathematical foundations such as finite series and vector math, offer promising solutions for effective and secure Private Information

Retrieval.

2.6 Commitment Scheme

Commitment Scheme is a cryptographic primitive that allows a user to securely lock in a value, preventing it from being modified or tampered with. Once the value is committed, it becomes binding and immutable, ensuring its integrity throughout the process.[7]
The Commitment Scheme involves a two-step process: commitment and reveal. During the commitment phase, the user generates a commitment, which is a cryptographic representation of the value. This commitment can be publicly shared without revealing the underlying value. Importantly, the commitment should be computationally infeasible to reverse-engineer to derive the original value.

In the future, during the reveal phase, the user can disclose the committed value, proving its consistency with the original commitment. This allows others to verify the integrity of the committed value and ensure that it has not been altered since the commitment was made.

The math behind Commitment Schemes depends on the scenario. Sometimes it's 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're ways to use logarithms/exponentiation, statistics, and bitwise operations to ensure the security of hidden values no matter the environment or strength of adversary.

2.7 Pseudo-random Number Generator

Pseudo-Random Number Generator is the method used to generate a sequence of numbers or symbols that appears to exhibit randomness within a cryptographic system. It is utilized in different contexts, including the generation of cryptographic keys, nonces (unique numbers used for one-time purposes), or temporary digital signatures.[7]
The math behind Pseudo-random Number Generators is broken down into 2 categories. There exist truly randomly generated numbers that 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 is a routing protocol that facilitates communication by utilizing proxy servers to transmit messages on behalf of clients. The key feature of Mix Networks is the mixing of messages received from multiple senders, sending them to their destinations in random order. This randomization makes it highly challenging to trace the digital trail of

communication.[7]

The mathematical principles behind Mix Networks involve the encryption of messages using a public key. Initially, the messages are encrypted by sealing them with the public key, and the destination address is appended to the cipher text. This concatenated data is then encrypted again using the public key, creating nested encryption. Only the proxy server's private key can unseal and decrypt both the destination address and the message, enabling it to forward the communication securely.

.

3 Secure Hash Algorithm 1

The acronym SHA stands for **Secure Hash Algorithm**, which represents a collection of one-way hash functions extensively utilized in computer science and mathematics. These algorithms exhibit all the characteristics previously described for one-way hash functions. The following section specifically delves into SHA-1 and the mathematical process by which a message is transformed into a hash.

3.1 How it Works

SHA-1 is designed to accept messages of any length and generate a 160-bit hash value. This hash value is typically represented as a hexadecimal number, consisting of 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) 0100000100100000010101000110010101110011011101001000000000000000
   0000000000000000000000000000000000000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000000

0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000

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) 0100000100100000010101000110010101110011011101001000000000000000
   0000000000000000000000000000000000000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000000
   0000000000000000 0000000000000000000000000000000000000000000000000
   0000000000000000110000
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

The SHA-1 process involves multiple steps that provide insights into why one-way hash functions possess their characteristic properties. In particular, Step 4 highlights how the output of SHA-1 is of a fixed length. By ensuring that the input string is congruent to 512, the data can be efficiently padded and divided, regardless of its original size. Additionally, Step 11 further reinforces the fixed length output by maintaining limited variable reassignment to the original five pre-defined variables.[5].

SHA-1 was widely employed in the mid-2000s but is now deemed insecure against adversaries with significant resources. It has been shown that computers with sufficient computational power can find collisions with a complexity of 2 to the power of 60. While brute-force collision search remains the only known method to break a Secure Hash Algorithm, SHA-1 is susceptible to chosen-prefix attacks. Nevertheless, SHA-1 was deliberately chosen to illustrate the workings of one-way hashing functions from a mathematical perspective.

To address the security concerns, the publishers of SHA-1 introduced SHA-2 and SHA-3 after extensive research and collaboration among mathematicians and computer scientists worldwide. SHA-2 is a family of Secure Hash Algorithms that provide fixed output bit lengths equivalent to powers of 2. On the other hand, SHA-3 is the latest addition to the algorithmic family and incorporates additional rounds of reassignment, more variables, words, and blocks compared to SHA-1. SHA-3 boasts the highest level of security against

collision attacks and length extension attacks, the latter being attacks on messages where the content does not need to be known.

It is important to note that the scale of strings and values worked with in Section 3.1 would be considerably larger and impractical to handle manually. Consequently, SHA-3 is replacing SHA-1 whenever possible, except in cases where the hashing algorithm is used for purposes such as data storage or execution efficiency rather than security. Both SHA-2 and SHA-3 are commonly employed in blockchain and network security applications due to their ability to form hashing merkle trees, where the accessibility and validity of data chunks depend on previous data chunks' authenticity and integrity.[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.