VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

CRYPTOGRPAHY AND NETWORK SECURITY
AAT REPORT
on

Analysis of SHA -1

Submitted by

Samir Kumar (1BM18CS091)
Rutazeet Ritik Rout (1BM18CS151)

Under the Guidance of
Prof. Lohith J J
Assistant Professor, BMSCE

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B. M. S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Mar-2021 to Jun-202

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the AAT work entitled "**Analysis of SHA-1**" is carried out by **Samir Kumar (1BM18CS091), and Rutazeet Ritik Rout (1BM18CS151)** who are bonafide students of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visveswaraya Technological University, Belgaum during the year 2021. The AAT report has been approved as it satisfies the academic requirements in respect of **Cryptography and Network Security (20CS6PCCNS)** work prescribed for the said degree.

Signature of the Guide	Signature of the HOD
Prof. Lohith J J	Dr. Umadevi V
Assistant Professor	Associate Prof. & Head, Dept. of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

B. M. S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, Samir Kumar (1BM18CS091) and Rutazeet Ritk Rout (1BM18CS151), students of 6th Semester, B.E, Department of Computer Science and Engineering, B.M.S. College of Engineering, Bangalore, here by declare that, this AAT entitled "Analysis of SHA-1" has been carried out by us under the guidance of Prof. Lohith J J, Assistant Professor, Department of CSE, B. M. S. College of Engineering, Bangalore during the academic semester Mar-2021-Jun-2021

We also declare that to the best of our knowledge and belief, the development reported here is not from part of any other report by any other students.

Signature

Samir Kumar (1BM18CS091)

Rutazeet Ritik Rout (1BM18CS151)

Chapter 1
Introduction

Problem Statement: -

Analyzing SHA-1 and demonstration of collision attack by CWI and Google against SHA-1, understanding SHA-256 and its role in cryptocurrencies like Bitcoin.

SHA-1 (**Secure Hash Algorithm-1**) is a cryptographic hash function which takes an input and produces a 160-bit (20-byte) hash value known as a message digest – typically rendered as a hexadecimal number, 40 digits long. It was designed by the United States National Security Agency, and is a U.S. Federal Information Processing Standard.
In February 2017, CWI Amsterdam and Google announced they had performed a collision attack against SHA-1, publishing two dissimilar PDF files which produced the same SHA-1 hash. As of 2020, chosen-prefix attacks against SHA-1 are practical. As such, it is recommended to remove SHA-1 from products as soon as possible and instead use SHA-256**.**

SHA-256 (**Secure Hash Algorithm-256**) is a cryptographic hash function designed by the United States National Security Agency (NSA) and first published in 2001. They are built using the Merkle–Damgård construction, from a one-way compression function itself built using the Davies–Meyer structure from a specialized block cipher. Currently, the best public attacks break preimage resistance for 52 out of 64 rounds of SHA-256, and collision resistance for 46 out of 64 rounds of SHA-256. SHA-256 thus play an important role in cryptocurrencies like Bitcoin where it is used to generate "**proof of work**" for each block in the block-chain.

Motivation: -

The fact that SHA is called Secure Hashing Algorithm but recently it was broken by collision attacks drew our attention towards cryptographic hashing algorithms, and the fact that better versions of this very algorithm are also now used in cryptocurrencies like Bitcoin motivated us to go through the details of these algorithms by referring several available research papers on internet and using software such as Cryptool to better understand the concepts in practice. We also designed our very own code to generate SHA-1 digest for any message using Java programming language.

Various Aspects: -

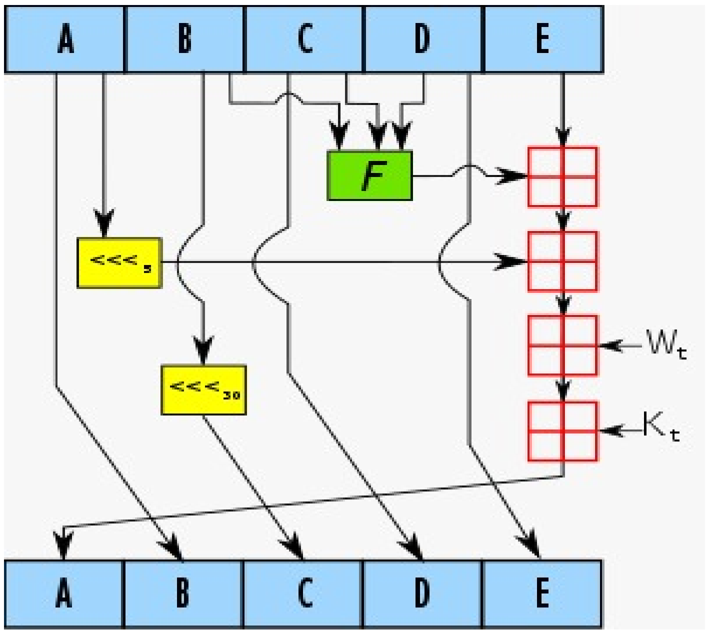
This industry cryptographic hash function standard is used for digital signatures and file integrity verification, and protects a wide spectrum of digital assets, including credit card transactions, electronic documents, open-source software repositories and software updates.
As of 2020, chosen-prefix attacks against SHA-1 are practical. As such, it is recommended to remove SHA-1 from products as soon as possible and instead use SHA-2 or SHA-3.
SHA-256 (a member of the SHA-2 cryptographic hash functions) is used in several different parts of the Bitcoin network:

- Mining uses SHA-256 as the Proof of work algorithm.
- SHA-256 is used in the creation of bitcoin addresses to improve security and privacy.

Chapter 2

Methodology

Step 1: Analysis of SHA-1



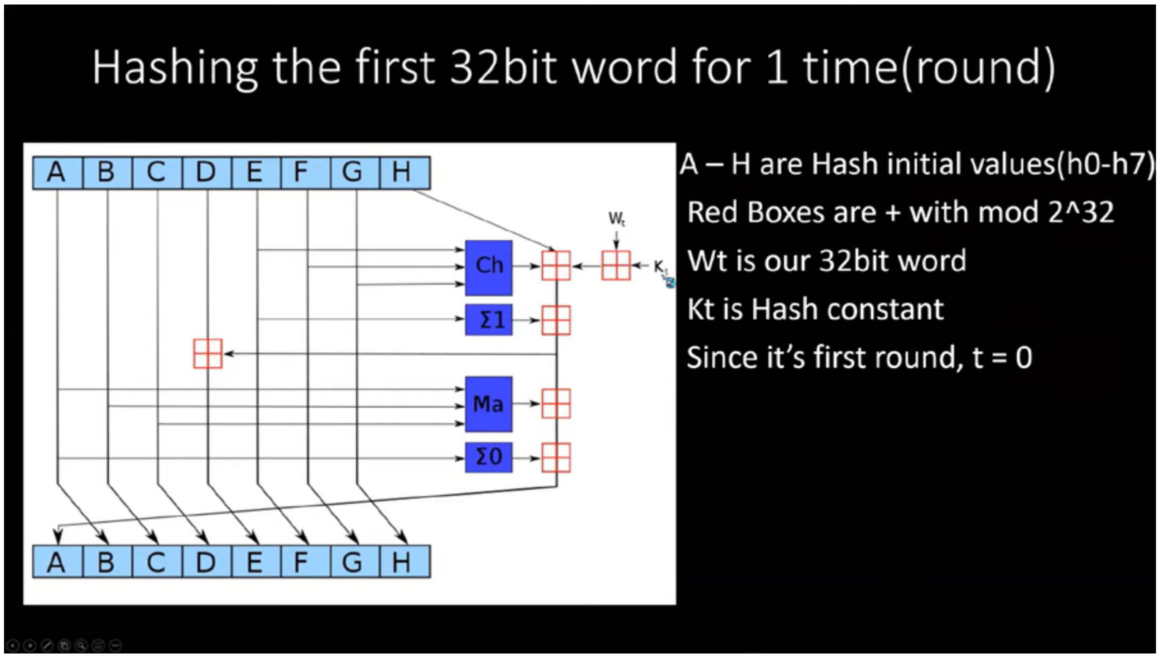
- SHA-1 or Secure Hash Algorithm 1 is a cryptographic hash function which takes an input and produces a 160-bit (20-byte) hash value. This hash value is known as a message digest.
- This message digest is usually then rendered as a hexadecimal number which is 40 digits long.
- SHA-1 forms part of several widely used security applications and protocols, including TLS and SSL, PGP, SSH, S/MIME, and IPsec.

Step 2: Collision-attack on SHA-1 using Cryptool

SHA-1 was believed to be a secure algorithm but since 2005 SHA-1 has not been considered secure against well-funded opponents; as of 2010 many organizations have recommended its replacement. In February 2017, CWI Amsterdam and Google announced they had performed a collision attack against SHA-1, publishing two dissimilar PDF files which produced the same SHA-1 hash.

We have demonstrated the same attack using Cryptool software using same PDF files as input i.e., shattered1.pdf and shattered2.pdf, we observe that both files produce same SHA-1 digest which proves the fact that SHA-1 remains no longer safe for use specially where it is used in digital signatures.

Step 3: Analysis of SHA-256



Hashing the first 32bit word for 1 time(round)

A – H are Hash initial values(h0-h7)

Red Boxes are + with mod 2^32

Wt is our 32bit word

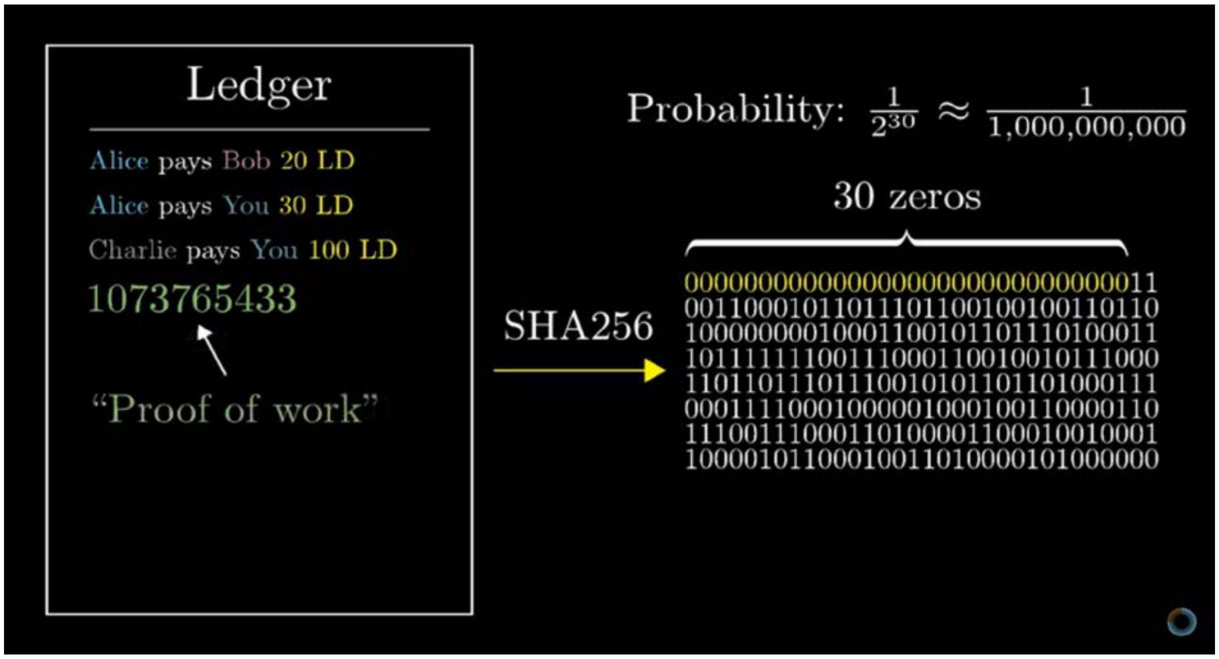
Kt is Hash constant

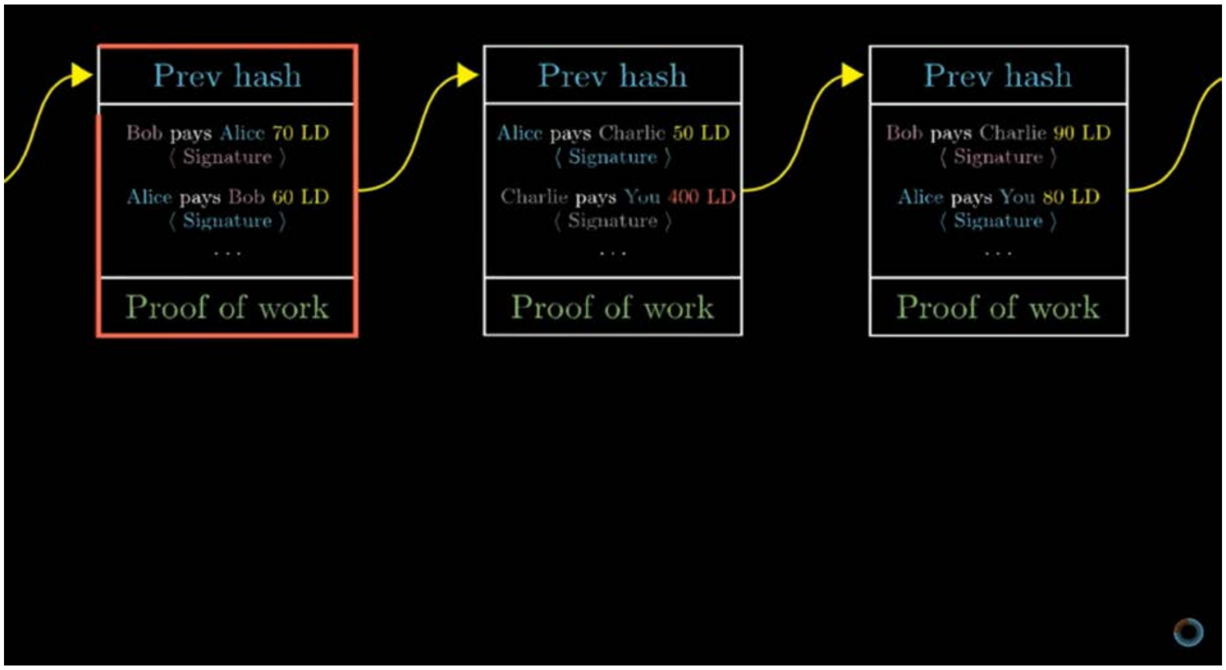
Since it's first round, t = 0

- A hash is not ‘encryption’ – it cannot be decrypted back to the original text (it is a ‘one-way’ cryptographic function, and is a fixed size for any size of source text). This makes it suitable when it is appropriate to compare ‘hashed’ versions of texts, as opposed to decrypting the text to obtain the original version.

- **SHA-256** (**Secure Hash Algorithm-256**) is a member of the SHA-2 cryptographic hash functions designed by the United States National Security Agency (NSA) and first published in 2001. They are built using the Merkle–Damgård construction, from a one-way compression function itself built using the Davies–Meyer structure from a specialized block cipher.

Step 4: Application of SHA-256 in Bitcoin





- A protocol which describes how to accept or reject transactions and in what order so that we can feel confident that anyone else in the world follows the same protocol and has a personal ledger that looks the same as ours, is addressed in the original Bitcoin paper. At a high level, the solution Bitcoin offers to trust whichever ledger has the most computational work put into it.

- Imagine someone shows a list of transactions, and they say “I found a special number so that when you put this number at the end of list of transactions, and apply SHA256 the entire thing, the first 30 bits of the output are zeros”.

- For a random message, the probability that the hash happens to start with 30 successive zeros is 1 in 2^30, which is about 1 in a billion. Because SHA256 is a cryptographic hash function, the only way to find a special number like this is just guessing and checking. And once we know the number, we can quickly verify that this hash really does start with 30 zeros. This is called “proof of work”.

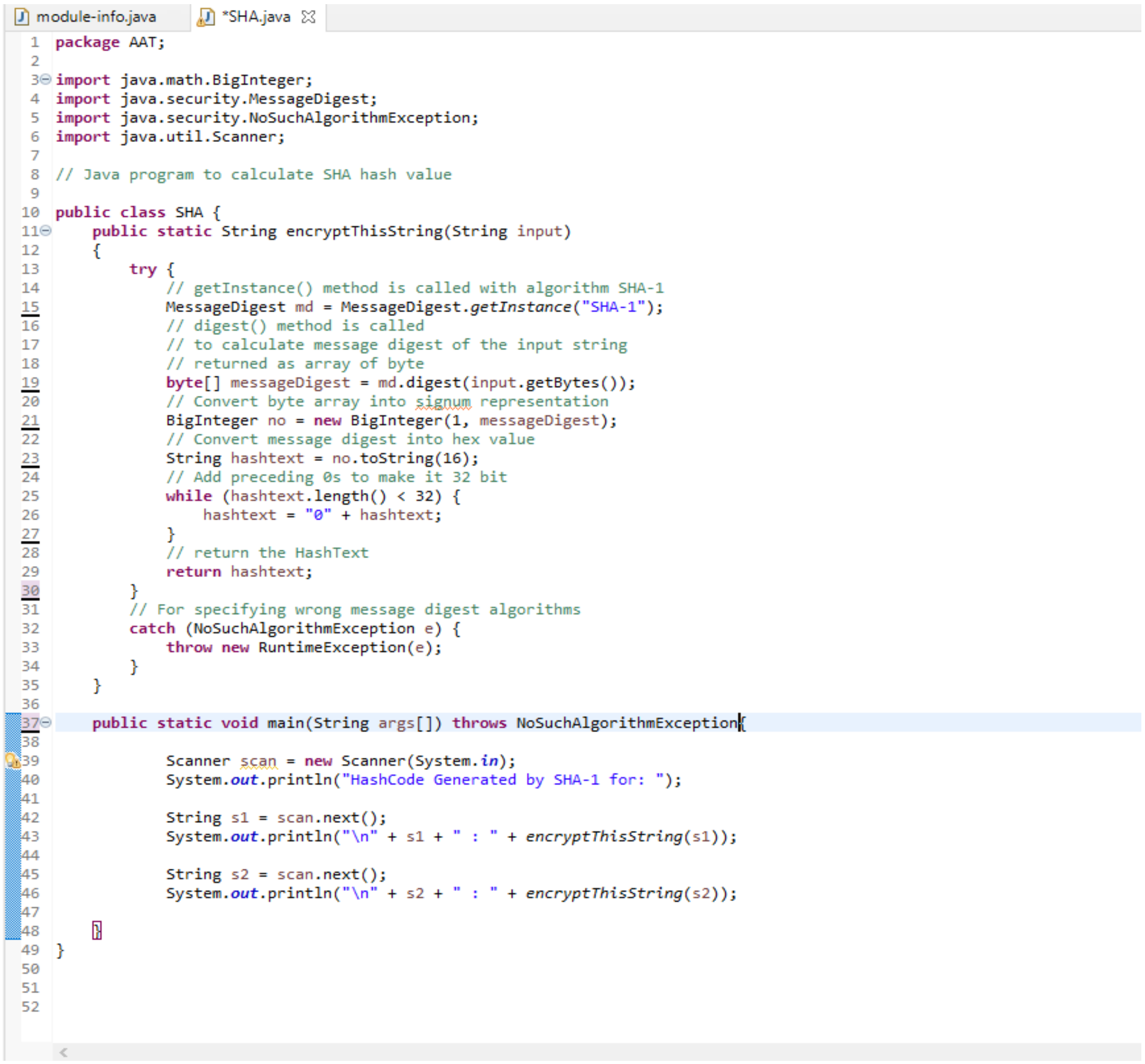
- Also, to make sure there is a standard way to order of these blocks, we’ll make it so that a block has to contain the hash of the previous block. That way, if you change any block, or try to swap the order of two blocks, it would change the block after it, which changes that block’s hash, which changes the next block, and so on. That would require redoing all the work, finding a new special number for each of these blocks that makes their hashes start with 30 zeros.

Implementing SHA- 1 using Java Language

- To calculate cryptographic hashing value in Java, MessageDigest Class is used, under the package java.security.

- This Algorithms are initialized in static method called getInstance(). After selecting the algorithm, it calculates the digest value and return the results in byte array.

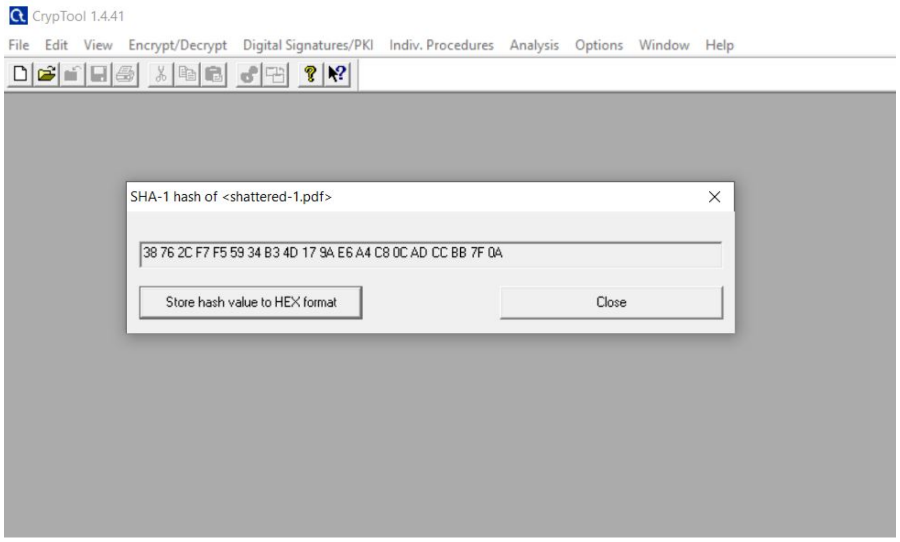
- BigInteger class is used, which converts the resultant byte array into its sign-magnitude representation. This representation is converted into hex format to get the MessageDigest



```java
package AAT;

import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Scanner;

// Java program to calculate SHA hash value

public class SHA {
    public static String encryptThisString(String input)
    {
        try {
            // getInstance() method is called with algorithm SHA-1
            MessageDigest md = MessageDigest.getInstance("SHA-1");
            // digest() method is called
            // to calculate message digest of the input string
            // returned as array of byte
            byte[] messageDigest = md.digest(input.getBytes());
            // Convert byte array into signum representation
            BigInteger no = new BigInteger(1, messageDigest);
            // Convert message digest into hex value
            String hashtext = no.toString(16);
            // Add preceding 0s to make it 32 bit
            while (hashtext.length() < 32) {
                hashtext = "0" + hashtext;
            }
            // return the HashText
            return hashtext;
        }
        // For specifying wrong message digest algorithms
        catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        }
    }

    public static void main(String args[]) throws NoSuchAlgorithmException{

            Scanner scan = new Scanner(System.in);
            System.out.println("HashCode Generated by SHA-1 for: ");

            String s1 = scan.next();
            System.out.println("\n" + s1 + " : " + encryptThisString(s1));

            String s2 = scan.next();
            System.out.println("\n" + s2 + " : " + encryptThisString(s2));

    }
}
```

Chapter 3

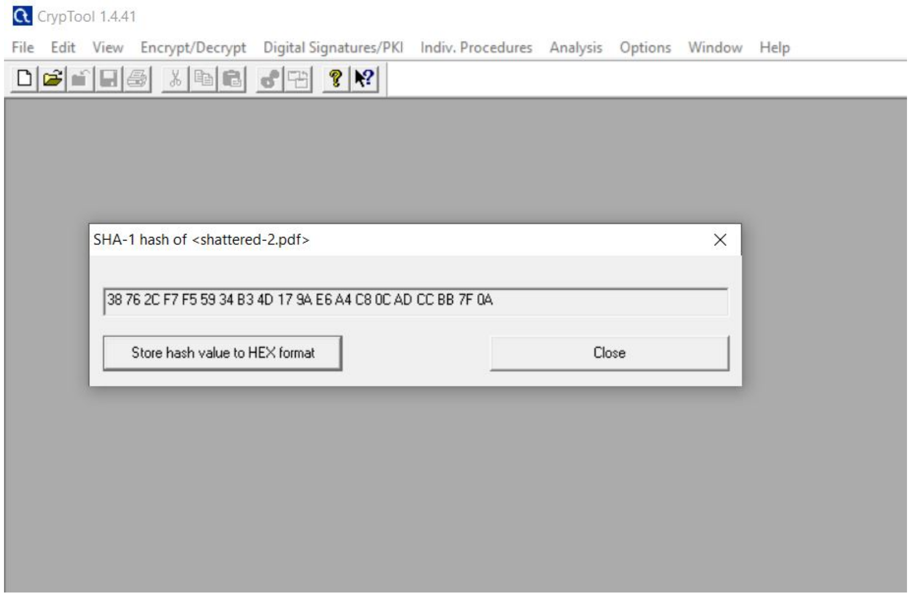
Results and Discussion


CrypTool 1.4.41
File Edit View Encrypt/Decrypt Digital Signatures/PKI Indiv. Procedures Analysis Options Window Help
Store hash value to HEX format
Close

SHA-1 hash of <shattered-1.pdf>

```
38 76 2C F7 F5 59 34 B3 4D 17 9A E6 A4 C8 0C AD CC BB 7F 0A
```

Fig. 1 – SHA-1 digest for shattered1.pdf

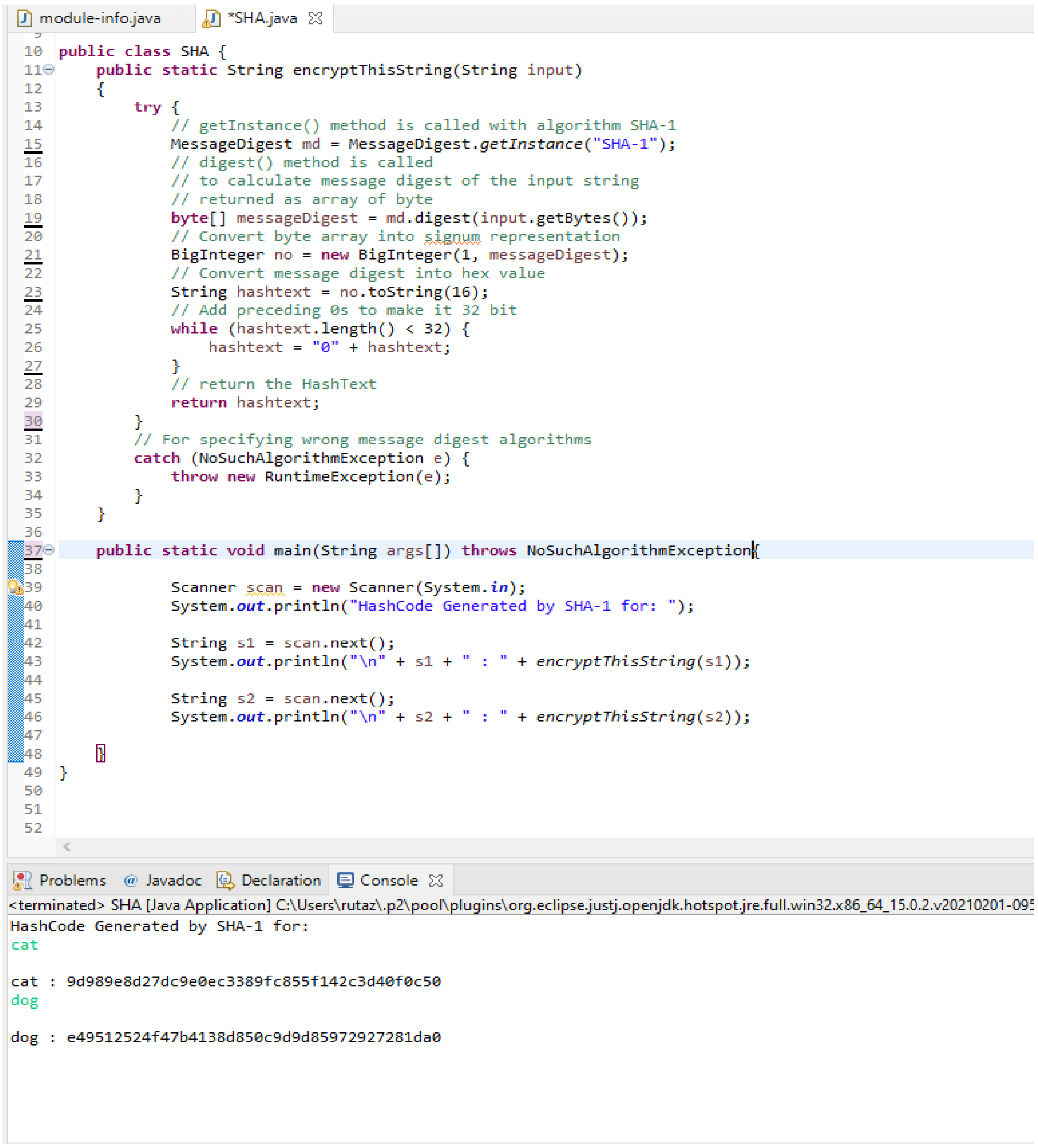

CrypTool 1.4.41
File Edit View Encrypt/Decrypt Digital Signatures/PKI Indiv. Procedures Analysis Options Window Help
Store hash value to HEX format
Close

SHA-1 hash of <shattered-2.pdf>

```
38 76 2C F7 F5 59 34 B3 4D 17 9A E6 A4 C8 0C AD CC BB 7F 0A
```

Fig. 2 – SHA-1 digest for shattered2.pdf

- Two different pdf files shattered1.pdf and shattered2.pdf generating same SHA-1 hash digest, first shown by CWI and Google in February 2017.

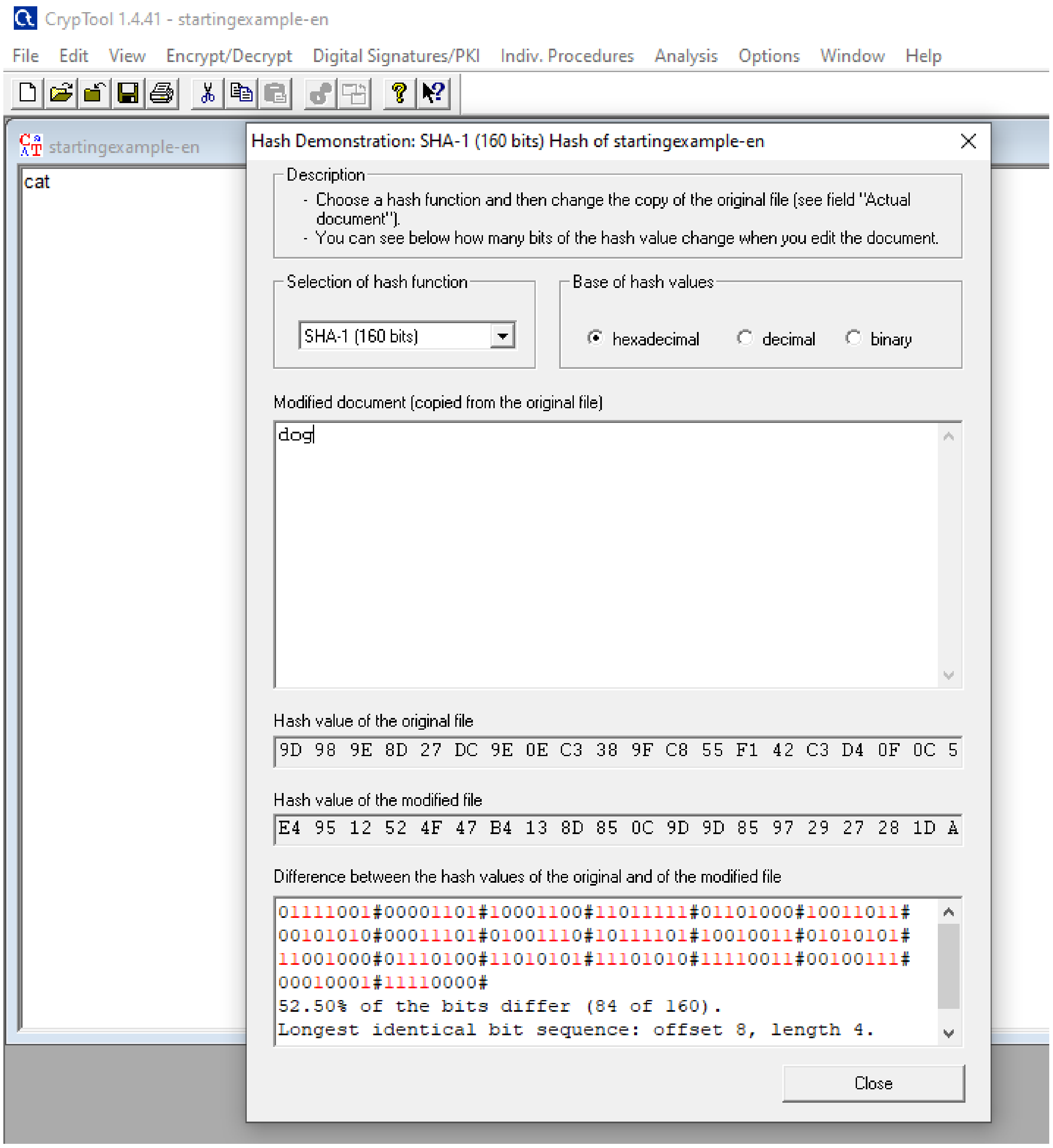


```java
public class SHA {
    public static String encryptThisString(String input)
    {
        try {
            // getInstance() method is called with algorithm SHA-1
            MessageDigest md = MessageDigest.getInstance("SHA-1");
            // digest() method is called
            // to calculate message digest of the input string
            // returned as array of byte
            byte[] messageDigest = md.digest(input.getBytes());
            // Convert byte array into signum representation
            BigInteger no = new BigInteger(1, messageDigest);
            // Convert message digest into hex value
            String hashtext = no.toString(16);
            // Add preceding 0s to make it 32 bit
            while (hashtext.length() < 32) {
                hashtext = "0" + hashtext;
            }
            // return the HashText
            return hashtext;
        }
        // For specifying wrong message digest algorithms
        catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        }
    }

    public static void main(String args[]) throws NoSuchAlgorithmException{

        Scanner scan = new Scanner(System.in);
        System.out.println("HashCode Generated by SHA-1 for: ");

        String s1 = scan.next();
        System.out.println("\n" + s1 + " : " + encryptThisString(s1));

        String s2 = scan.next();
        System.out.println("\n" + s2 + " : " + encryptThisString(s2));

    }
}
```

```
<terminated> SHA [Java Application] C:\Users\rutaz\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-095
HashCode Generated by SHA-1 for: 
cat

cat : 9d989e8d27dc9e0ec3389fc855f142c3d40f0c50
dog

dog : e49512524f47b4138d850c9d9d85972927281da0
```

Fig. 3 – Code for SHA-1 in Java Language

- The screenshot shown above shows the output generated by the code for the following input as follows:
  cat: 9d989e8d27dc9e0ec3389fc855f142c3d40f0c50
  dog: e49512524f47b4138d850c9d9d85972927281da0


CrypTool 1.4.41 - startingexample-en
File Edit View Encrypt/Decrypt Digital Signatures/PKI Indiv. Procedures Analysis Options Window Help
startingexample-en
Close

cat

Hash Demonstration: SHA-1 (160 bits) Hash of startingexample-en

Description
- Choose a hash function and then change the copy of the original file (see field "Actual document").
- You can see below how many bits of the hash value change when you edit the document.

Selection of hash function: SHA-1 (160 bits)

Base of hash values: [✓ hexadecimal] decimal binary

Modified document (copied from the original file)

```
dog
```

Hash value of the original file

```
9D 98 9E 8D 27 DC 9E 0E C3 38 9F C8 55 F1 42 C3 D4 0F 0C 5
```

Hash value of the modified file

```
E4 95 12 52 4F 47 B4 13 8D 85 0C 9D 9D 85 97 29 27 28 1D A
```

Difference between the hash values of the original and of the modified file

```
01111001#00001101#10001100#11011111#01101000#10011011#
00101010#00011101#01001110#10111101#10010011#01010101#
11001000#01110100#11010101#11101010#11110011#00100111#
00010001#11110000#
52.50% of the bits differ (84 of 160).
Longest identical bit sequence: offset 8, length 4.
```

Fig. 4 – Verification of output generated using Cryptool

- The screenshot shown above generates the same output as generated by the code, hence this verifies that our code is correct.

  cat: 9d989e8d27dc9e0ec3389fc855f142c3d40f0c50
  dog: e49512524f47b4138d850c9d9d85972927281da0

Chapter 4

Conclusion and Future Work

We learned how SHA-1 and SHA-256 works in practice and saw how SHA-1 produced the same digest for two different files and hence we conclude that we can no longer use SHA-1 for generating digital signatures in particular since it demands high level of privacy and security which now cannot be guaranteed through SHA-1 which is now prone to collision attacks.

We also learned how SHA-256 (a member of the SHA-2 cryptographic hash functions) is used in several different parts of the Bitcoin network:

- Mining uses SHA-256 as the Proof of work algorithm.
- SHA-256 is used in the creation of bitcoin addresses to improve security and privacy.

Also writing code for SHA-1 in Java programming language helped us to better understand the algorithm and we could generate the hash digest for any message using our code, we also verified our results through Cryptool software.

We would like to extend our code in future so that it is able to generate hash-digest for different file types such as pdf files, text files, word documents, etc. This would really expand the scope of our work and make our code versatile and complete in true sense.

References:

1. Design for High Throughput SHA-1 Hash Function on FPGA: Jae-woon Kim, Hu-ung Lee, Youjip Won

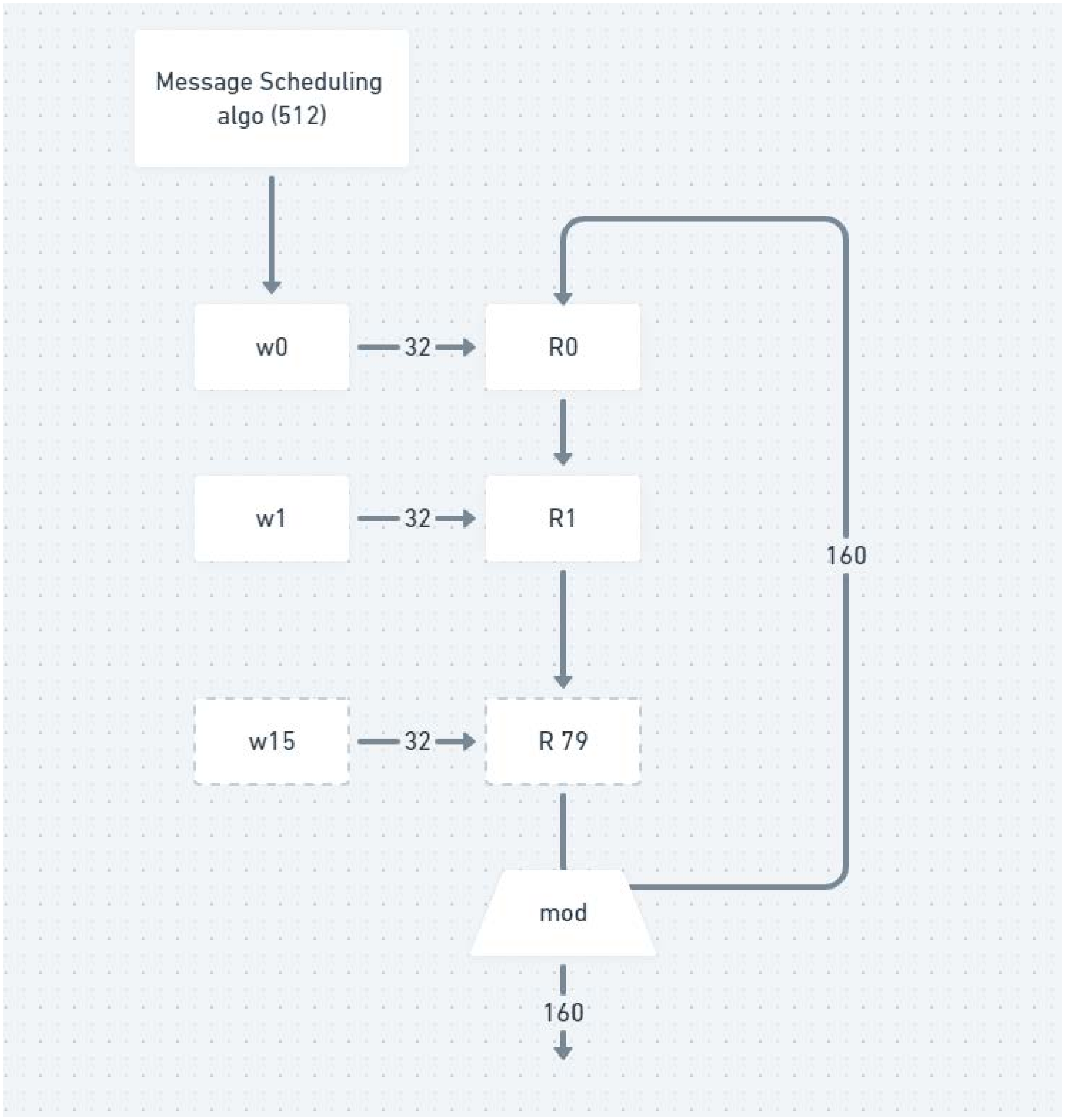
2. The first collision for full SHA-1: Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, Yarik Markov

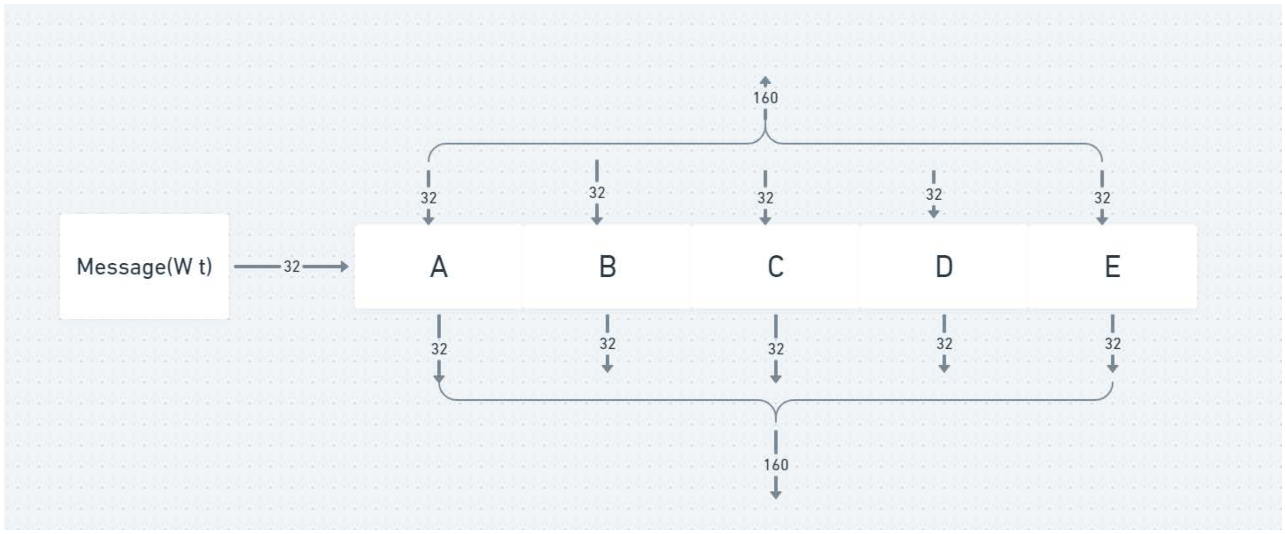
3. Hardware Complexity of SHA-1 and SHA-256 Based on Area and Time Analysis: Jun Cheol Jeon and Kang-Joong Seo, Kee-Won Kim

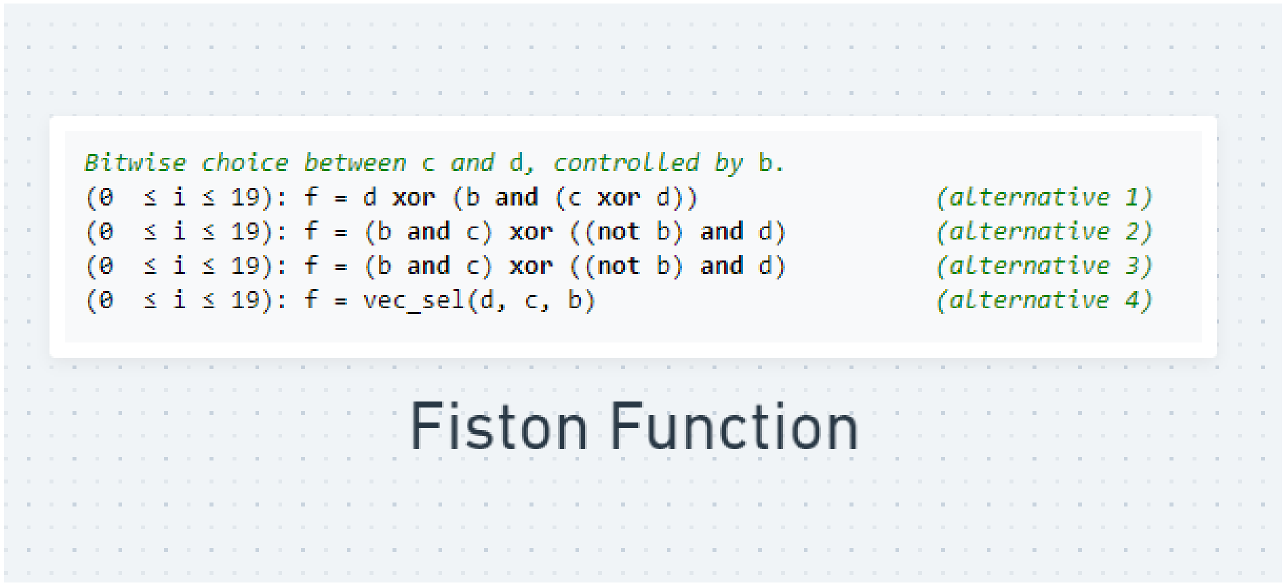
4. Implementation of Efficient SHA-256 Hash Algorithm for Secure Vehicle Communication using FPGA: Chanbok Jeong and Youngmin Kim

5. Bitcoin: A Peer-to-Peer Electronic Cash System: Satoshi Nakamoto

https://whimsical.com/8Lpq2BGJL448FZ1BtgzxLd - explanation of algorithm and the function

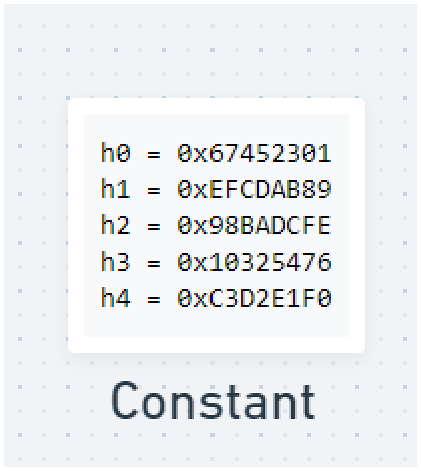

Message Scheduling algo (512)
w0
32
R0
w1
32
R1
w15
32
R 79
160
mod
160





```
Bitwise choice between c and d, controlled by b.
(0  ≤ i ≤ 19): f = d xor (b and (c xor d))                (alternative 1)
(0  ≤ i ≤ 19): f = (b and c) xor ((not b) and d)          (alternative 2)
(0  ≤ i ≤ 19): f = (b and c) xor ((not b) and d)          (alternative 3)
(0  ≤ i ≤ 19): f = vec_sel(d, c, b)                       (alternative 4)
```

Fiston Function



```
h0 = 0x67452301
h1 = 0xEFCDAB89
h2 = 0x98BADCFE
h3 = 0x10325476
h4 = 0xC3D2E1F0
```

Constant