CryptoProject

Cryptography Project

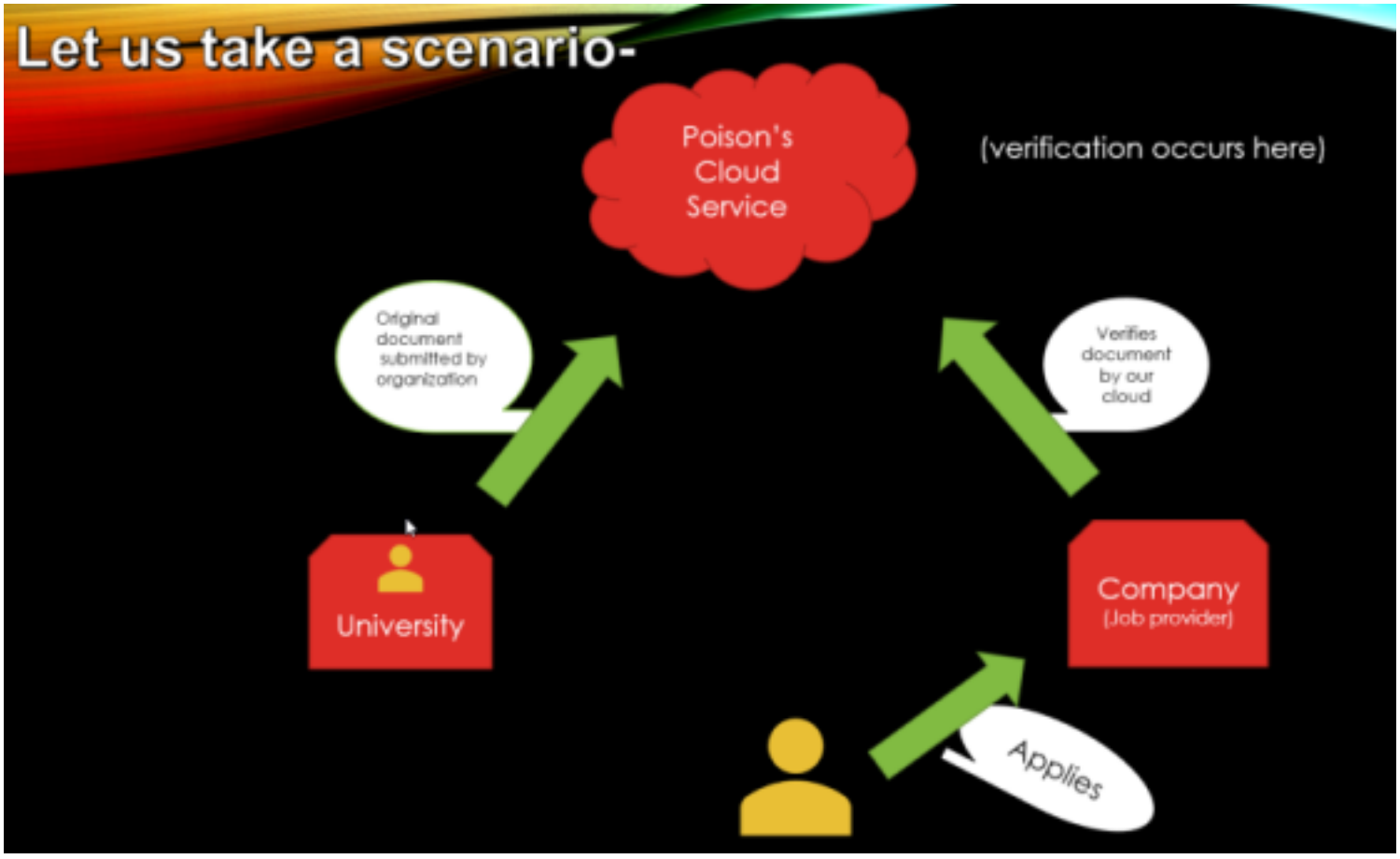
Created by :- **Hritish Kumar**

This was my 1st Project. So there can be mistakes and i've also tested this algorithm it works like a charm.

Totally Secure Hashing Algorithm(T-SHA)

Abstract/Problem statement:- Doctoring of documents have been quite common these days. I Proposed a method to verify fake documents using cryptography techniques and designed and modified original SHA-1 algorithm to make it collision free and harder to Brute Force.

Idea Behind It :-

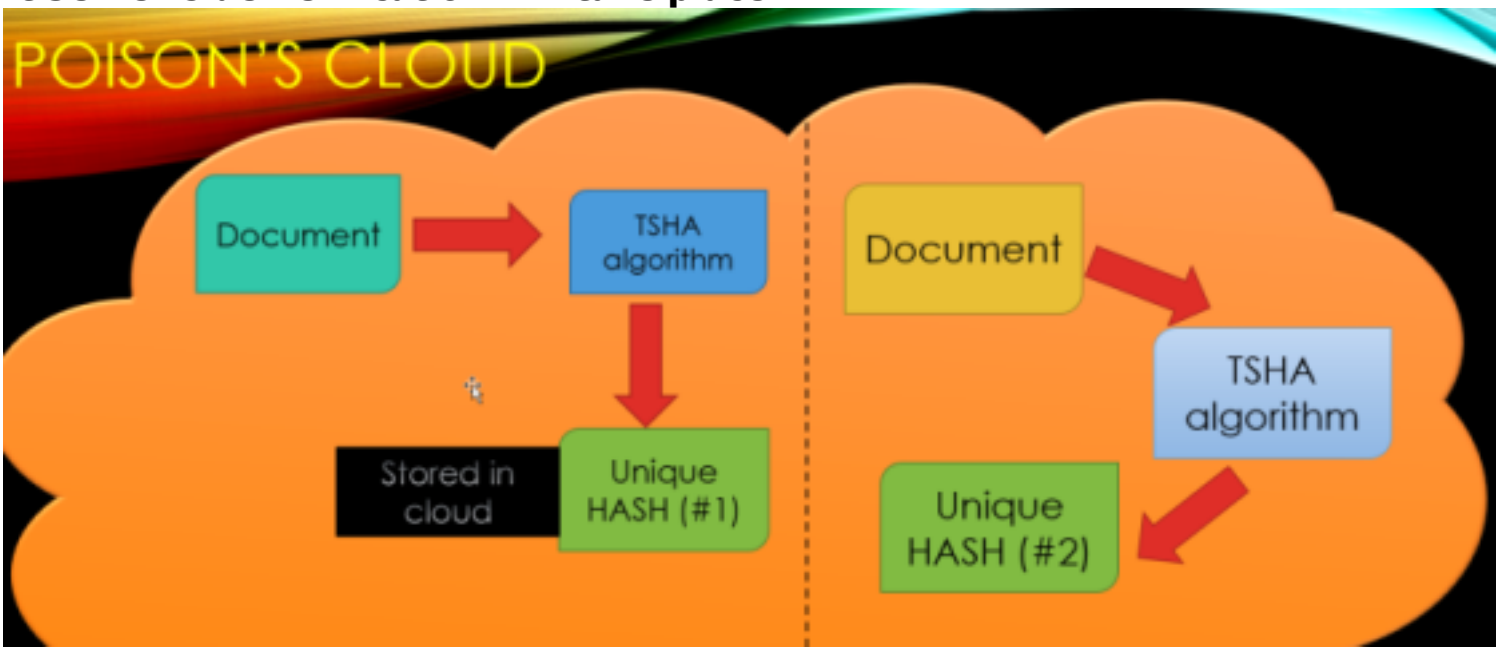


In the above scenario, A Organization (university) will register with cloud services and upload the original certification of a person now we will sign a hash using TSHA and store it on our cloud.

Now if a person applies for a job at a company . Then the company can come to our

website and verify the document by uploading the same document that person provided them.

At Server-side Verification will take place.



This figure represents the server side view.
Here on the left side we already have legit file hash stored and on the right side the user will upload a document to verify if the file is authentic or not.

Introduction

There are a lot of hashing algorithm but when it comes to safety and security of our clients we don't compromise. So my team has created a Hashing algorithim T-SHA

there were a lot of problems now a days in hashing algorithm so we've created a fast, secured and collosion free algorithm. Read the documentation below for understanding about the T- SHA

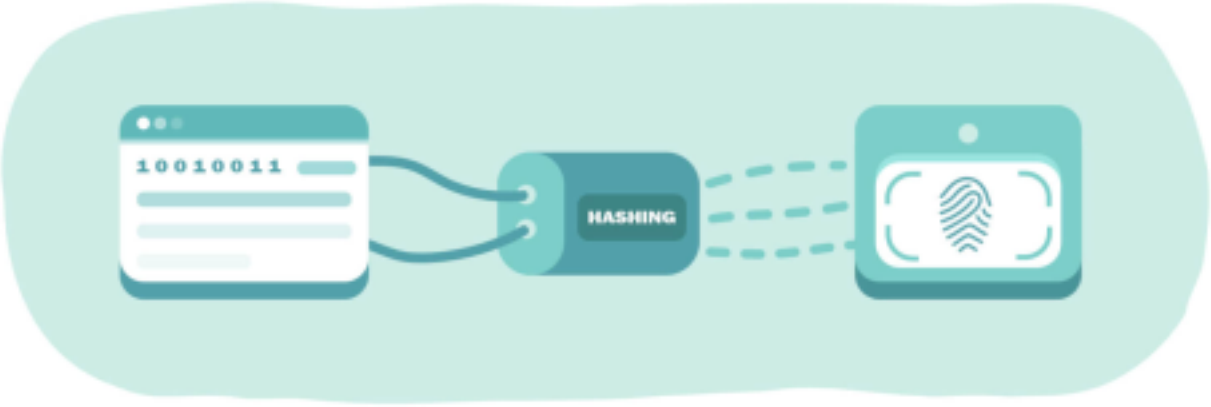
and collosion in hashing algorithm**.**

A collision occurs when two distinct pieces of data—a document, a binary, or a website's certificate—hash to the same digest as shown above.

In practice, collisions should never occur for secure hash functions. However if the hash algorithm has some flaws, as SHA-1 does, a well-funded attacker can craft a collision.

The attacker could then use this collision to deceive systems that rely on hashes into accepting a malicious file in place of its benign counterpart.

For example, two insurance contracts with drastically different terms.



We have made an advanced, more secure, faster and collosion free hashing algorithim. It uses fiestal cipher, sha1, XOR and some other custom operations for hashing and we've also tested the hashing algorithm with similar 10000 messages.

What are the benefits of Hashing?

One main use of hashing is to compare two files for equality.

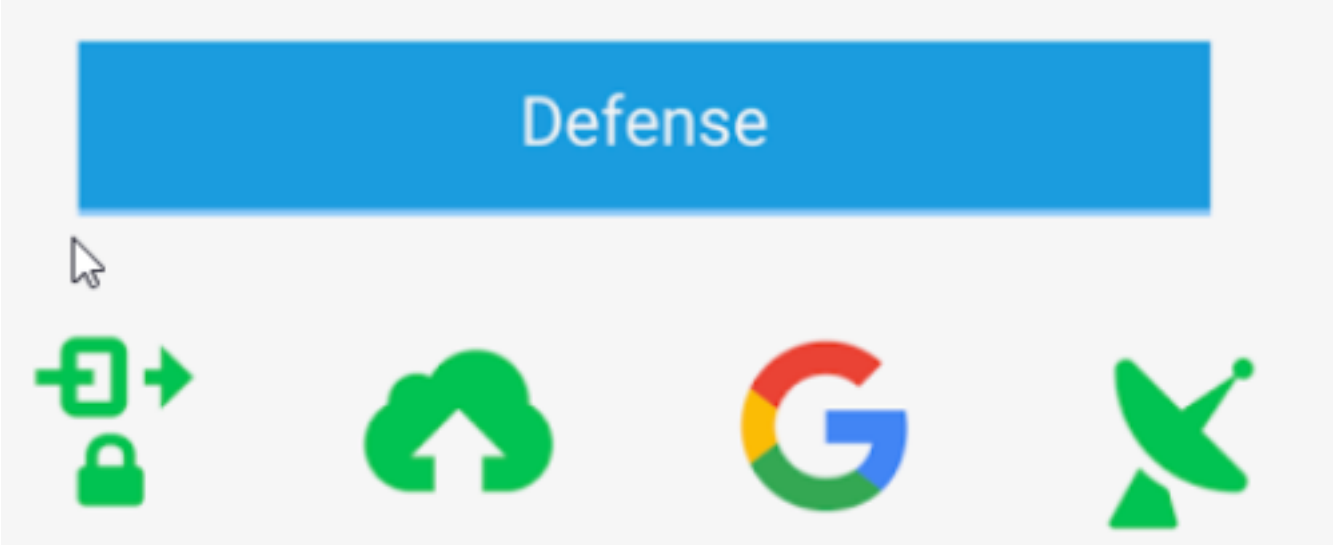
Without opening two document files to compare them word-for-word, the calculated hash values of these files will allow the owner to know immediately if they are different.

Hashing is also used to verify the integrity of a file after it has been transferred from one place to another, typically in a file backup program like SyncBack.

To ensure the transferred file is not corrupted, a user can compare the hash value of both files. If they are the same, then the transferred file is an identical copy.

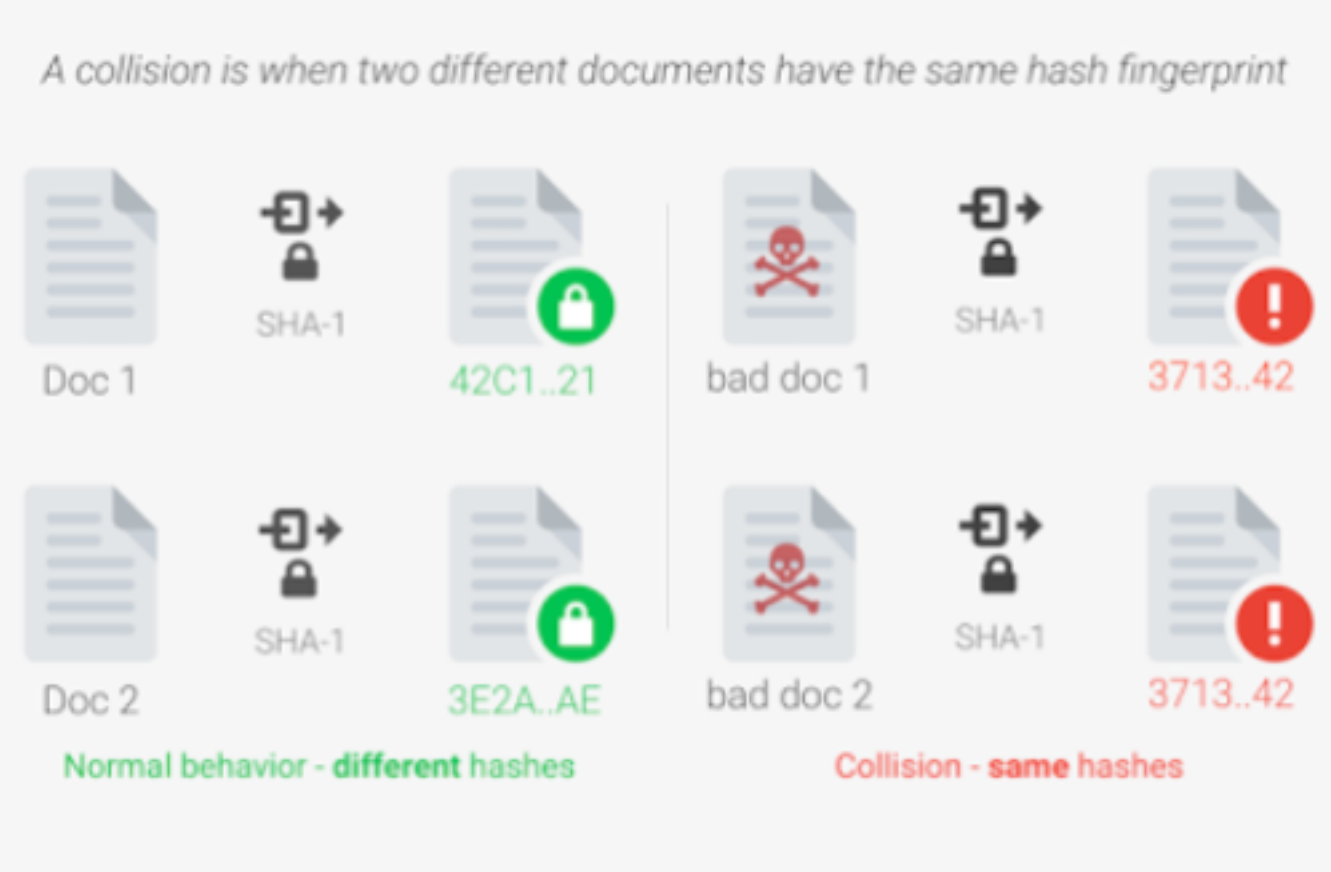
In some situations, an encrypted file may be designed to never change the file size nor the last modification date and time (for example, virtual drive container files).

In such cases, it would be impossible to tell at a glance if two similar files are different or not, but the hash values would easily tell these files apart if they are different.



Our algorithm satisfy all the Properties of Hash algorithm:-

- Running the same hash function on the same input data must yield the same hash value.
- Small changes to input data should result in large changes to the hash value.
- Each resultant hash value for different input data should be unique.
- The hashing process must be one way (i.e. it can't be reversed).



Survey with 10+ Papers

Sha-1

On February 23, 2017 Google announced the first SHA1 collision.

Announcing the first SHA1 collision,this requires a lot of computing power and resources. Since this never occurred naturally in real world under normal conditions we can rule out security risks today but not tomorrow.

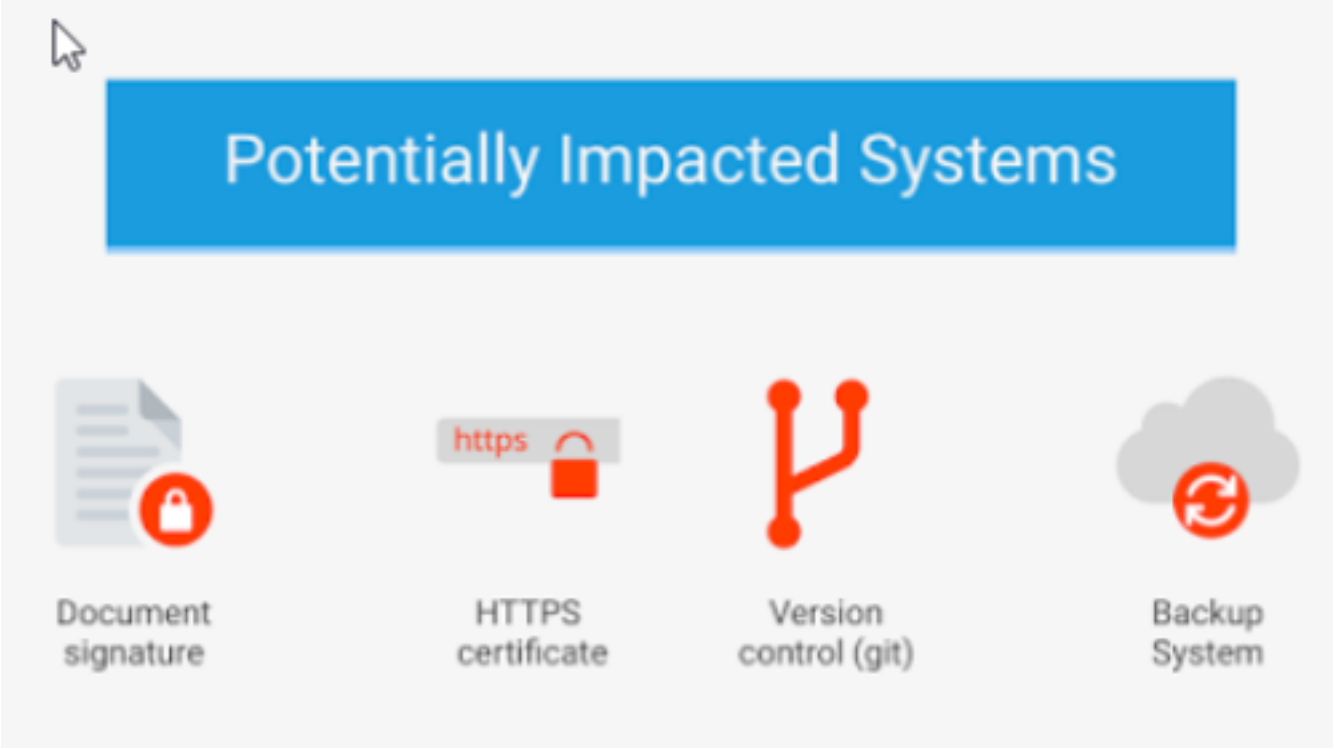
Cryptographic hash functions like SHA-1 are a cryptographer's swiss army knife. You'll find that hashes play a role in browser security, managing code repositories, or even just detecting duplicate files in storage. Hash functions compress large amounts of data into a small message digest. As a cryptographic requirement for wide-spread use, finding two messages that lead to the same digest should be computationally infeasible. Over time however, this requirement can fail due to attacks on the mathematical underpinnings of hash functions or to increases in computational power.

Today, more than 20 years after of SHA-1 was first introduced, Google announced the first practical technique for generating a collision. This represents the culmination of two years of research that sprung from a collaboration between the CWI Institute in Amsterdam and Google. We've summarized how we went about generating a collision

below. As a proof of the attack, we are releasing two PDFs that have identical SHA-1 hashes but different content.

For the tech community, our findings emphasize the necessity of sunsetting SHA-1 usage. Google has advocated the deprecation of SHA-1 for many years, particularly when it comes to signing TLS certificates. As early as 2014, the Chrome team announced that they would gradually phase out using SHA-1. We hope our practical attack on SHA-1 will cement that the protocol should no longer be considered secure.

We hope that our practical attack against SHA-1 will finally convince the industry that it is urgent to move to safer alternatives such as SHA-256.



Finding the SHA-1 collision

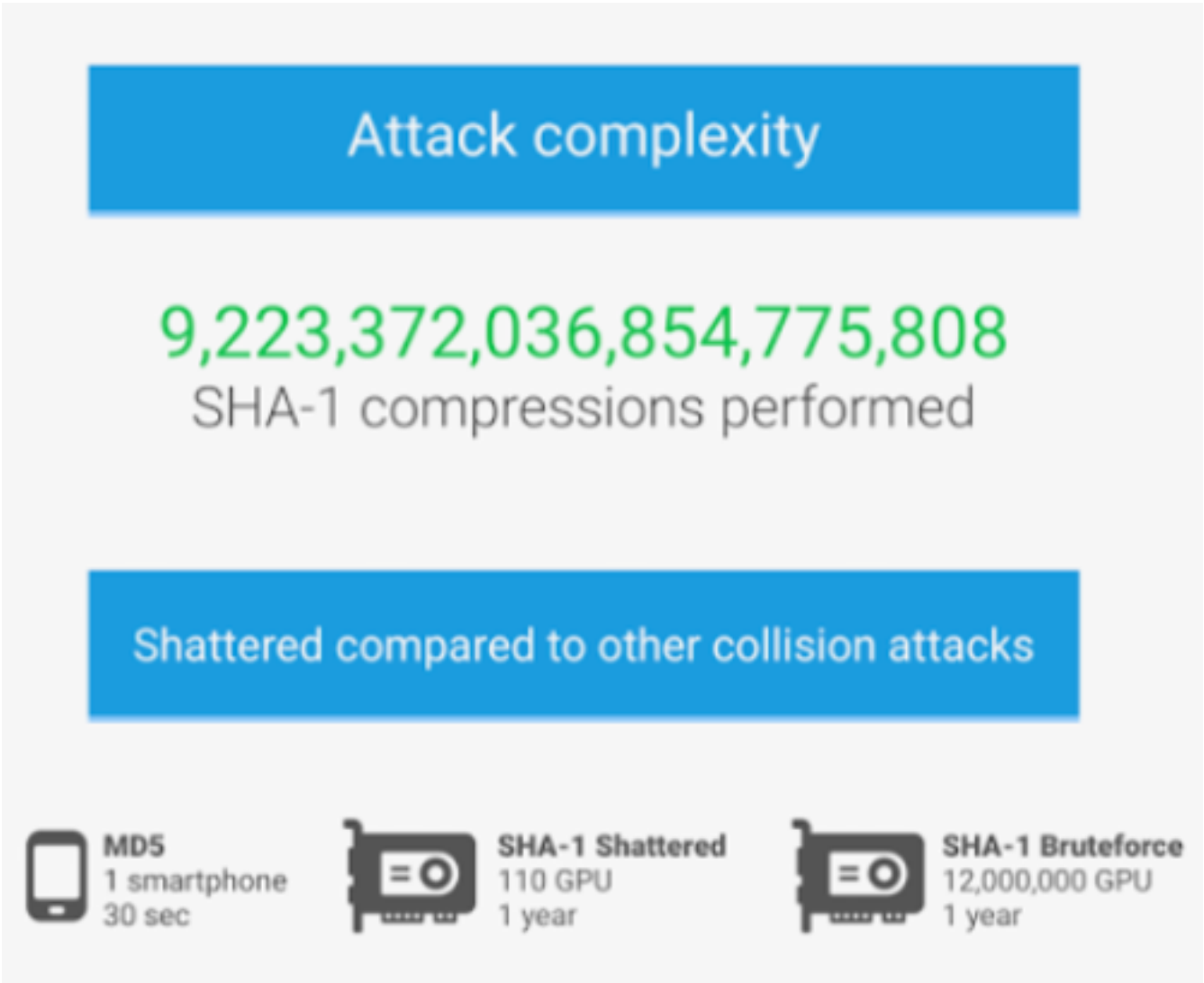
In 2013, Marc Stevens published a paper that outlined a theoretical approach to create a SHA-1 collision. We started by creating a PDF prefix specifically crafted to allow us to generate two documents with arbitrary distinct visual contents, but that would hash to the same SHA-1 digest. In building this theoretical attack in practice we had to overcome some new challenges. We then leveraged Google's technical expertise and cloud infrastructure to compute the collision which is one of the largest computations ever completed.

Here are some numbers that give a sense of how large scale this computation was:

Nine quintillion (9,223,372,036,854,775,808) SHA1 computations in total

6,500 years of CPU computation to complete the attack first phase

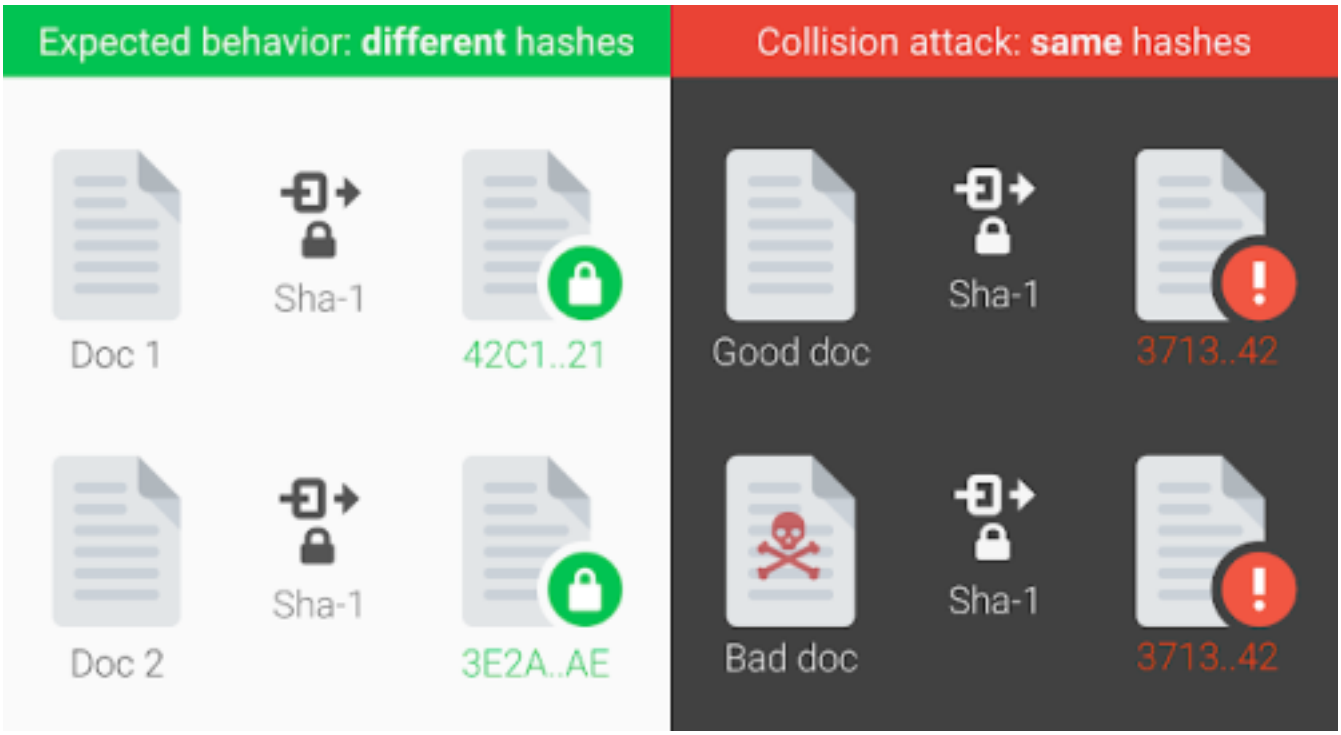
110 years of GPU computation to complete the second phase



While those numbers seem very large, the SHA-1 shattered attack is still more than 100,000 times faster than a brute force attack which remains impractical.

Mitigating the risk of SHA-1 collision attacks

Moving forward, it's more urgent than ever for security practitioners to migrate to safer cryptographic hashes such as SHA-256 and SHA-3. Following Google's vulnerability disclosure policy, we will wait 90 days before releasing code that allows anyone to create a pair of PDFs that hash to the same SHA-1 sum given two distinct images with some pre-conditions. In order to prevent this attack from active use, we've added protections for Gmail and GSuite users that detects our PDF collision technique. Furthermore, we are providing a free detection system to the public.



What is Collosion Resistance?

Collision resistance is a property of cryptographic hash functions:

A hash function H is collision resistant if it is hard to find two inputs that hash to the same output; that is, two inputs a and b such that H(a) = H(b), and a ≠ b.

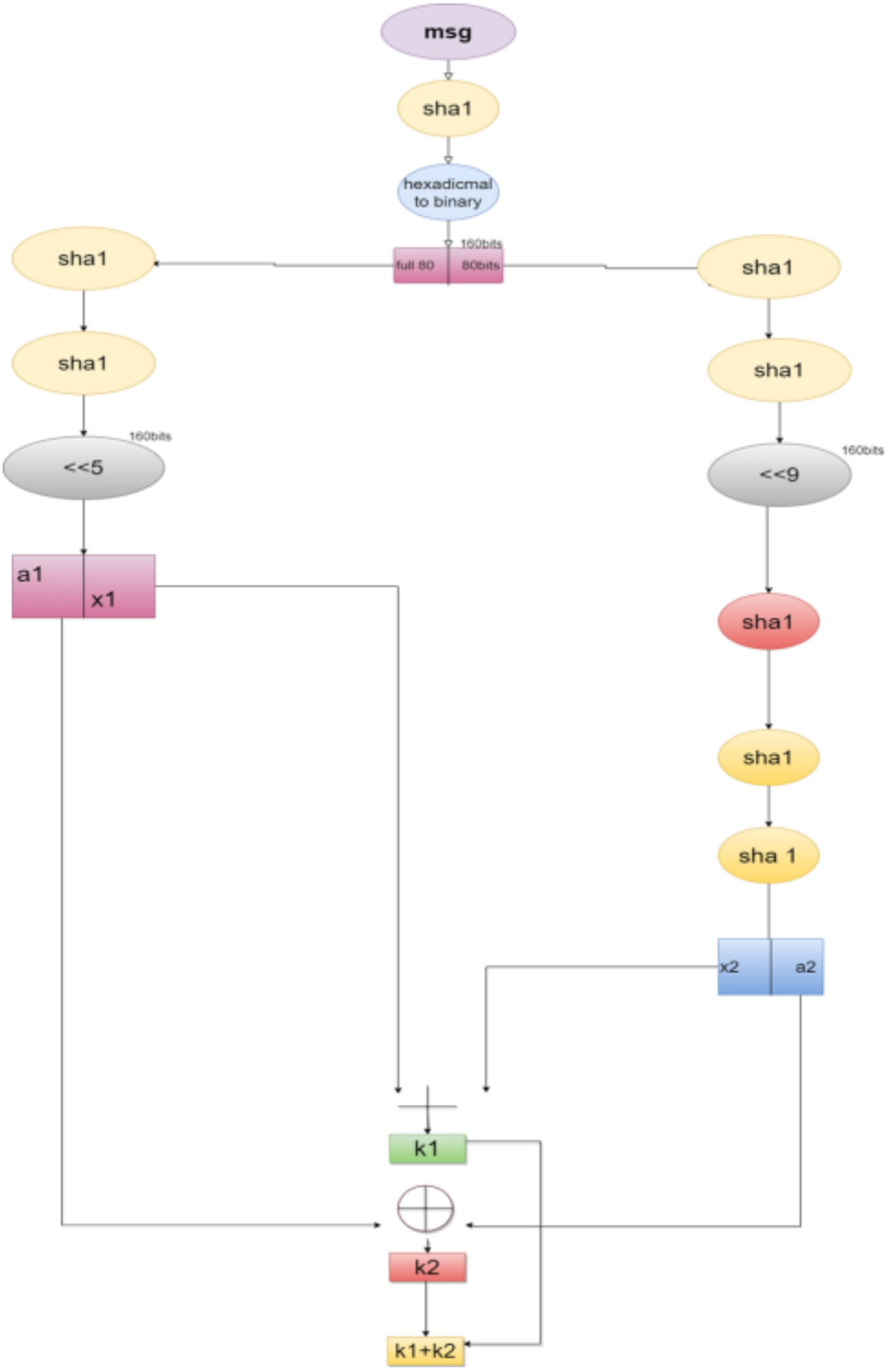
Every hash function with more inputs than outputs will necessarily have collisions.Consider a hash function such as SHA-256 that produces 256 bits of output from a large input

(≤ 264-1 bits). Since it must generate one of 2256 outputs for each member of a much larger set of inputs, the pigeonhole principle guarantees that some inputs will hash to the same output. Collision resistance does not mean that no collisions exist; simply that they are hard to find.

The "birthday paradox" places an upper bound on collision resistance: if a hash function produces N bits of output, an attacker who computes only 2N/2 hash operations on random input is likely to find two matching outputs. If there is an easier method than this brute-force attack, it is typically considered a flaw in the hash function.

Cryptographic hash functions are usually designed to be collision resistant. But many hash functions that were once thought to be collision resistant were later broken. MD5 and SHA-1 in particular both have published techniques more efficient than brute force for finding collisions.However, some hash functions have a proof that finding collisions is at least as difficult as some hard mathematical problem (such as integer factorization or discrete logarithm). Those functions are called provably secure.

Making of T-SHA



Overview: I've implemented Fiestal Ciphers,XOR, OR & AND operations , Circularshifting , Padding and a-lot of things.

Detailed:
Cryptographic functions designed to keep data secured. It works by transforming the data using a hash function: an algorithm that consists of bitwise operations, modular additions, and compression functions. The hash function then produces a fixed-size string that looks nothing like the original. These algorithms are designed to be one-way functions, meaning that once they're transformed into their respective hash values, it's virtually impossible to transform them back into the original data. There are a total of 160 rounds in TSHA.

A common application of TSHA is to encrypting passwords, as the server-side only needs to keep track of a specific user's hash value, rather than the actual password. This is helpful in case an attacker hacks the database, as they will only find the hashed functions and not the actual passwords, so if they were to input the hashed value as a password, the hash function will convert it into another string and subsequently deny access.
An arbitrary constant of any length is given as input. The msg is passed to sha1 which gives an output of 160 bits or convert it to 160 bits in the hexadecimal form which is then converted to the binary form of 160 bits. It is then divided into two parts left and right of 80 bits each. The left 80 bit is passed to sha1 twice and converted to 160 bits and is circular shifted 5 bits. This is divided into two 80 bits of named a1 and x1. The right part is passed to sha1 twice and is circularly shifted by 9 bits and passed thrice through sha1. This is divided into two 80 bits of named a2 and x2.
x1 and x2 are passed through and function which gives an output k1. a1 and a2 is passed through and function which gives an output k2. K1 and k2 are appended together. the given output will go through the same procedure 160 times and the final output will become our hash.

Merits of T-SHA:-

- Collision resistance,
- Immpossible to Crack,
- Bruteforce is next to impossible due to complexity and the time taken will be very high.

How to use the algorithm and verify for collosion?

I've made a web application for demonstration of the TSHA , Implemented conversion of text to hash and **file signing & verifying** Functions.

After unziping the file you can use any text editor to view the code and we've used php , html , css , js , python to build our project.

Zip file contains all the files related to the project.

Requirements:

- Internet
- Chrome
- Netlify **Web Hosting Services** for upload sigining and verifying services.

Steps:

1. Unzip the file
2. Open Index.html
3. Follow the webpage

Code for checking collosion:-

```
import time as wait

try:
    def TSHA(TEXT):

        def sha1(data):
            bytes = ""

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

            for n in range(len(data)):
                bytes+='{0:08b}'.format(ord(data[n]))
            bits = bytes+"1"
            pBits = bits
            while len(pBits)%512 != 448:
                pBits+="0"
            pBits+='{0:064b}'.format(len(bits)-1)

            def chunks(l, n):
                return [l[i:i+n] for i in range(0, len(l), n)]

            def rol(n, b):
                return ((n << b) | (n >> (32 - b))) & 0xffffffff

            for c in chunks(pBits, 512):
                words = chunks(c, 32)
                w = [0]*80
                for n in range(0, 16):
                    w[n] = int(words[n], 2)
                for i in range(16, 80):
                    w[i] = rol((w[i-3] ^ w[i-8] ^ w[i-14] ^ w[i-16]), 1)

                a = h0
                b = h1
                c = h2
                d = h3
```

```
        e = h4


        for i in range(0, 80):
            if 0 <= i <= 19:
                f = (b & c) | ((~b) & d)
                k = 0x5A827999
            elif 20 <= i <= 39:
                f = b ^ c ^ d
                k = 0x6ED9EBA1
            elif 40 <= i <= 59:
                f = (b & c) | (b & d) | (c & d)
                k = 0x8F1BBCDC
            elif 60 <= i <= 79:
                f = b ^ c ^ d
                k = 0xCA62C1D6

            temp = rol(a, 5) + f + e + k + w[i] & 0xffffffff
            e = d
            d = c
            c = rol(b, 30)
            b = a
            a = temp

        h0 = h0 + a & 0xffffffff
        h1 = h1 + b & 0xffffffff
        h2 = h2 + c & 0xffffffff
        h3 = h3 + d & 0xffffffff
        h4 = h4 + e & 0xffffffff

    return '%08x%08x%08x%08x%08x' % (h0, h1, h2, h3, h4)

def AND(r1,x1):
    result = ''
    for i in range(len(r1)):
        if r1[i]=='1' and x1[i]=='1':
            result=result+'1'
        if r1[i] == '0' and x1[i] == '1':
            result = result + '0'
        if r1[i] == '1' and x1[i] == '0':
            result = result + '0'
        if r1[i] == '0' and x1[i] == '0':
            result = result + '0'

    return result
def hexadecimalTObinary(n):
    res = "{0:08b}".format(int(n, 16))
```

```
    return res.zfill(160)

def binaryToDecimal(n):
    return int(n, 2)

def decimal_to_binary(value):
    k=''.join(format(ord(i),'b')for i in value)
    return k

def XOR(r1,x1):
    result = ''
    for i in range(len(r1)-1):
        if r1[i]=='1' and x1[i]=='1':
            result=result+'0'
        if r1[i] == '0' and x1[i] == '1':
            result = result + '1'
        if r1[i] == '1' and x1[i] == '0':
            result = result + '1'
        if r1[i] == '0' and x1[i] == '0':
            result = result + '0'
    return result


def cirularshift(hash,r_no):
    y=hash[0:r_no]
    x=hash[r_no+1:]
    x=x+y
    return x

def f1(hash) :
    x = sha1(sha1(hash))
    y = cirularshift(x,5)
    return hexadecimalTObinary(y)

def f2(hash) :
    x = sha1(sha1(hash))
    y = cirularshift(x, 9)
    y2 = sha1(sha1(sha1(y)))
    return hexadecimalTObinary(y2)


def round(message):
    full=hexadecimalTObinary(sha1(message))
    l=  full[:80]
    r1=f2(l)
    a1=r1[0:80]
    x1=r1[80:]
```

```
        r=full[80:]
        l1=f1(r1)
        a2=l1[80:]
        x2=l1[:80]
        k1=AND(a1,a2)
        k2=XOR(x1,x2)
        return k1+k2

    def finalHash(TEXT):
        gen_hashes=[]
        gen_hashes.append(round(TEXT))
        i=0
        while i<=160:
            j=round(gen_hashes[-1])
            gen_hashes.append(j)
            i=i+1
        return gen_hashes[-1]
    return sha1(finalHash(TEXT))

def generatingOfTestingString(l):
    testing_string=[]
    for i in range(0,l+1):
        y='a'*(i)+'@'+'a'*(l-i)
        testing_string.append(y)
    return testing_string

l=generatingOfTestingString(int(input("Enter the number of Hashes you want to
verify: ")))

print("[+] Generating Hashes....\n")
for i in l:
    print(l.index(i),TSHA(i))
    print()

def checkingForCollosion(list):
    vault_keys=[]
    for i in list:
        if i not in vault_keys:
            vault_keys.append(i)
    print("[+] Processing.... \n")
    wait.sleep(2)
    print("[+] Checking for Collision now\n")
    wait.sleep(1)
    print("[+] This process may take a while...\n")
    wait.sleep(3)
    if len(list)==len(vault_keys):
        print("[+] No Collision Found!")
```

```
        else:
            print("[-] Collision Found!")

    checkingForCollosion(l)
except(Exception):
    print("\n\n[-] Invalid Input")
    exit("[-] Exiting Program!!!")
```

Code for Converting Text to Hash:-

```
try:
    def TSHA(TEXT):
        def sha1(data):
            bytes = ""

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

            for n in range(len(data)):
                bytes += '{0:08b}'.format(ord(data[n]))
            bits = bytes + "1"
            pBits = bits
            while len(pBits) % 512 != 448:
                pBits += "0"
            pBits += '{0:064b}'.format(len(bits) - 1)

            def chunks(l, n):
                return [l[i:i + n] for i in range(0, len(l), n)]

            def rol(n, b):
                return ((n << b) | (n >> (32 - b))) & 0xffffffff

            for c in chunks(pBits, 512):
                words = chunks(c, 32)
                w = [0] * 80
                for n in range(0, 16):
                    w[n] = int(words[n], 2)
                for i in range(16, 80):
                    w[i] = rol((w[i - 3] ^ w[i - 8] ^ w[i - 14] ^ w[i - 16]), 1)

                a = h0
                b = h1
                c = h2
                d = h3
```

```
        e = h4

        for i in range(0, 80):
            if 0 <= i <= 19:
                f = (b & c) | ((~b) & d)
                k = 0x5A827999
            elif 20 <= i <= 39:
                f = b ^ c ^ d
                k = 0x6ED9EBA1
            elif 40 <= i <= 59:
                f = (b & c) | (b & d) | (c & d)
                k = 0x8F1BBCDC
            elif 60 <= i <= 79:
                f = b ^ c ^ d
                k = 0xCA62C1D6

            temp = rol(a, 5) + f + e + k + w[i] & 0xffffffff
            e = d
            d = c
            c = rol(b, 30)
            b = a
            a = temp

        h0 = h0 + a & 0xffffffff
        h1 = h1 + b & 0xffffffff
        h2 = h2 + c & 0xffffffff
        h3 = h3 + d & 0xffffffff
        h4 = h4 + e & 0xffffffff

    return '%08x%08x%08x%08x%08x' % (h0, h1, h2, h3, h4)

def AND(r1, x1):
    result = ''
    for i in range(len(r1)):
        if r1[i] == '1' and x1[i] == '1':
            result = result + '1'
        if r1[i] == '0' and x1[i] == '1':
            result = result + '0'
        if r1[i] == '1' and x1[i] == '0':
            result = result + '0'
        if r1[i] == '0' and x1[i] == '0':
            result = result + '0'

    return result

def hexadecimalTObinary(n):
    res = "{0:08b}".format(int(n, 16))
```

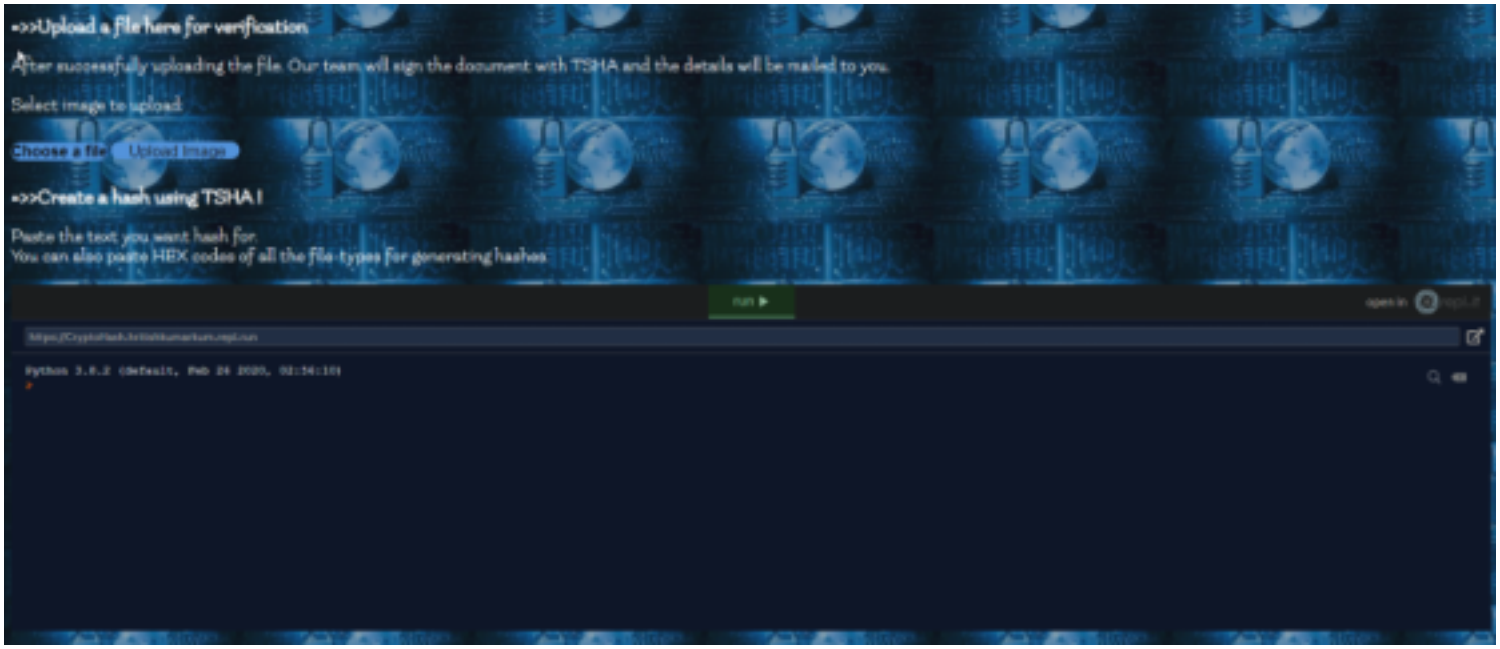
```
    return res.zfill(160)

def binaryToDecimal(n):
    return int(n, 2)

def decimal_to_binary(value):
    k = ''.join(format(ord(i), 'b') for i in value)
    return k

def XOR(r1, x1):
    result = ''
    for i in range(len(r1) - 1):
        if r1[i] == '1' and x1[i] == '1':
            result = result + '0'
        if r1[i] == '0' and x1[i] == '1':
            result = result + '1'
        if r1[i] == '1' and x1[i] == '0':
            result = result + '1'
        if r1[i] == '0' and x1[i] == '0':
            result = result + '0'
    return result

def cirularshift(hash, r_no):
    y = hash[0:r_no]
    x = hash[r_no + 1:]
    x = x + y
    return x

def f1(hash):
    x = sha1(sha1(hash))
    y = cirularshift(x, 5)
    return hexadecimalTObinary(y)

def f2(hash):
    x = sha1(sha1(hash))
    y = cirularshift(x, 9)
    y2 = sha1(sha1(sha1(y)))
    return hexadecimalTObinary(y2)

def round(message):
    full = hexadecimalTObinary(sha1(message))
    l = full[:80]
    r1 = f2(l)
    a1 = r1[0:80]
    x1 = r1[80:]
    r = full[80:]
    l1 = f1(r1)
```

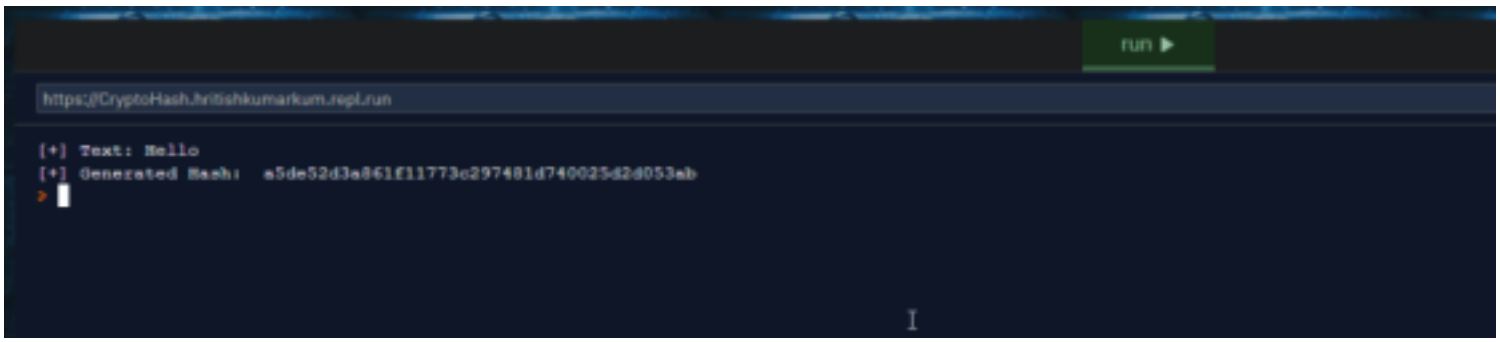
```
            a2 = l1[80:]
            x2 = l1[:80]
            k1 = AND(a1, a2)
            k2 = XOR(x1, x2)
            return k1 + k2

        def finalHash(TEXT):
            gen_hashes = []
            gen_hashes.append(round(TEXT))
            i = 0
            while i <= 160:
                j = round(gen_hashes[-1])
                gen_hashes.append(j)
                i = i + 1
            return gen_hashes[-1]

        return sha1(finalHash(TEXT))
    print("[+] Generated Hash: ",TSHA(input("[+] Text: ")))

except(Exception):
    print("\n\n[-] Invaild Input\n")
    exit("[-]Exiting Program")
```

Website:

I've use PHP,HTML & CSS for making website.
Pressing Run button will start the program.

I've attached some of screenshots of the website below for POC.





References:-

Marc Stevens (CWI Amsterdam),
Elie Bursztein (Google),
Pierre Karpman (CWI Amsterdam),
Ange Albertini (Google),
Yarik Markov (Google),

Alex Petit Bianco (Google),
Clement Baisse (Google),
Check their blogs for more info
https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html

For more info and project files mail at Hritish42@gmail.com

Created by Hritish & Aena