CSCE 4050/5050

Applications of Cryptography

Programming Project 2 –

Collision search using birthday attack

By:

Group 4

Shashidhar Kalapatapu – 11545863

Dharmateja Kollipara - 11555465

Project Task:

The objective of this project is to write a program implementing the generic birthday attack to find a collision. More specifically, the program runs the birthday attack against the BadHash44 function.

Birthday Attack

A Birthday attack is a cryptography attack based on the birthday paradox in probability theory. In probability theory, the birthday paradox means there is a good probability of two randomly selected people having the same birthday. In the context of cryptography, this concept is exploited between two or more parties. The attack has a fixed number of pigeonholes or permutations and the probability of collision increases with an increase in random attack attempts [1].

Collision search using Birthday Attack

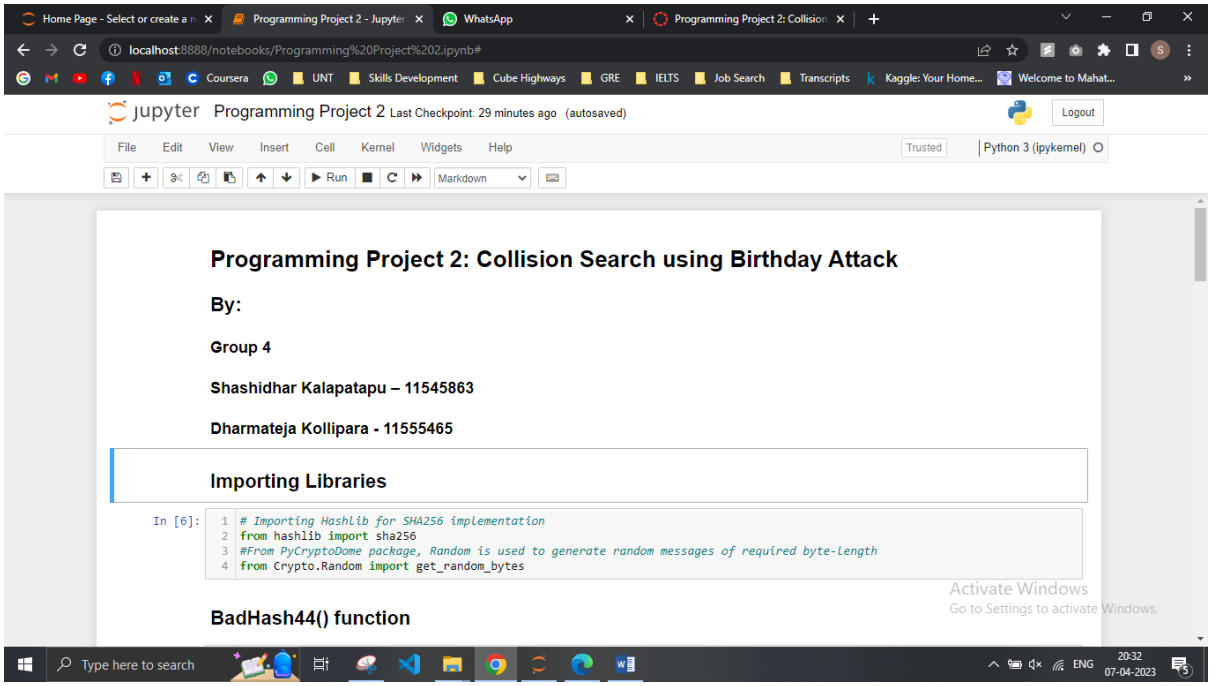
For our project, BadHash44() is a hash function that intentionally returns a small hash output generated using SHA256. It takes a random 256-bit input message, computes the SHA256 hash value, and returns only the first 44 bits of the hash. We will be finding a collision from BadHash44 by continuously giving it random input values.

The two tasks that will be done in this project are:

1. Search for collision in BadHash44 by finding two random inputs when given to BadHash44 as input, outputs the same hash value. When a collision is found, the two input messages and the corresponding bad hash value are reported.
2. Compute the hashes for the random messages in a table, sort the table on the hash value, and store them in a “hash.txt” or a “hash.csv” file. The BadHash44 function is run against all the random messages from the messages list.

Program Description:

For this project, the algorithm is written in Python 3.9.7 and run in the Anaconda Jupyter Notebook environment.



To run the attack, we only need to import and install two Python libraries.

- Hashlib – From this library, we are importing the sha256 method which generates a 256-bit Hash using the SHA256 algorithm.
- Crypto.Random – From this library, we are importing get_random_bytes() which can generate a random bytes object of the required length.

Importing Libraries

In [6]:

```
# Importing Hashlib for SHA256 implementation
from hashlib import sha256
#From PyCryptoDome package, Random is used to generate random messages of required byte-length
from Crypto.Random import get_random_bytes
```

Next, we will implement the BadHash44() function. This function takes a 256-bit random message (in Bytes) as the input. Using the imported SHA256 algorithm, it computes a 256-bit hash value in a hexadecimal Python string. But it finally returns only the first 44 bits of the hexadecimal string. Since each digit in hexadecimal represents 4 bits, 44 bits would be 11 digits in hexadecimal value.

BadHash44() function

```
]:  1  def BadHash44(bytesMessage):
    2      ''' Returns the first 44 bits of the SHA256 generated Hash.
    3      Input: 256-bit Message
    4      Output: First 44 bits of the SHA256 hash generated from the input bytes message
    5      '''
    6      # Get the SHA256 hash in hexadecimal string using sha256().hexdigest()
    7      hashByteString = sha256(bytesMessage).hexdigest()
    8      # Return the first 44 bits of the hexadecimal string.
    9      # Since each digit in hexadecimal value holds 4 bits, to get the 44 bits, we return the first 11 characters/digits
   10      return hashByteString[:11] # returns 44 bits hash
```

startBirthdayAttack() method

This function initiates a birthday attack on the BadHash44 function. We define 3 constants with “collisionFound”, “restartCounter” and the “messagesCount”. The “collisionFound” is a Boolean flag to check whether a collision has happened or not. The restartCounter is to track how many times the algorithm was restarted after not finding a collision. The messagesCount has the total number of random messages which are randomly generated every iteration.

startBirthdayAttack() function

```
 1  def startBirthdayAttack():
 2      '''
 3      Initiates Birthday attack on the BadHash44 function
 4      Eventually a collision happens in O(2^n/2) runtime
 5      '''
 6      collisionFound = False # Boolean flag to check for collisions.
 7      restartCounter = 0 # Counter to note how many times the generic algorithm was restarted
 8      messagesCount = 2**22 # Total Number of Random Messages which is generated
 9
10      print("[START] Starting Birthday Attack on BadHash44 Hash function ...\n")
11      print("[INFO] Total Random Messages to be generated : ", messagesCount)
12      print("[INFO] Current Program Restart Count: ", restartCounter)
13
14      #Search until collision is found
15      while not collisionFound:
16          hashList = []
17
18          print("[INFO] Computing {} 256-bit Random Messages and respective 44-bit Hash value ...".format(messagesCount))
19          for i in range(messagesCount):
20              inputMessage = get_random_bytes(32) # 256-bits or 32 bytes random message
21              hashOutput = BadHash44(inputMessage) # Call the badhash44 function here
22              hashList.append([hashOutput, inputMessage.hex()])
23          hashList.sort(key = lambda i: i[0])
```


Since the output is 44 bits, N = 44. We will generate 2^N/2 random messages. That would be 2^44/2 = **2^22 = 4,194,304** Random Messages generated every iteration.

Once the constants are defined, we use a loop where our generic algorithm runs. In this main loop, the steps are as follows:

- Declare an empty list that will store the (Hash Value, Input Message) pairs.
- In a second loop, generate 4,194,304 Random Messages using the get_random_bytes(32) function which returns a 32-byte or 256-bit Random Message.
- For each random message generated, use the BadHash44 function to compute its 44-bit Hash value. Append all the (Hash Value, Input Message) as pairs into the hash list.

After all the pairs are generated, this nested list is sorted on the value on the first part of each element. i.e. using a lambda function, this variable gets sorted on the Hash value as the key. Once we have a sorted list of hash and input pairs, we write it to a hash.txt file.

To check for collisions in the hash values, the function then compares adjacent elements in the sorted list. We use a loop to go through all the values in the hash list and check their adjacent value. As the list is sorted, if a collision exists, it is right beside the actual hash as a duplicate key. If a collision is found, the function terminates and prints the input messages that produced the colliding hash values.

If a collision is not found, we increment the restartCounter and the algorithm is repeated until collision is found.

```
        #write hashes to file
        print("[INFO] Writing Hash and Message pair to hash.txt file ...")
        myList = ["{0}: {1} \n".format(x1, x2) for (x1, x2) in hashList]
        with open('hash.txt', 'w') as out_file:
            out_file.writelines(myList)
        out_file.close()

        #Search for collisions in the sorted hash list
        print("[INFO] Searching for Collisions ...")
        for i in range(len(hashList) - 1):
            #If collision found, print messages, bad hash and break the loop
            if hashList[i][0] == hashList[i + 1][0]:
                print("[INFO] Collision Found!")
                collisionFound = True

                m1 = hashList[i][1]
                m2 = hashList[i + 1][1]
                badHash = hashList[i][0]
                print("[INFO] Collision on this Hash value: ", badHash)
                print("[INFO] Collision occured for these two input messages: ")
                print("[INFO] M1: ", m1)
                print("[INFO] M2: ", m2)

                print("[END] Total Program Restarts: ", restartCounter )
                return m1, m2, badHash

        #if collision not found continue the loop
        if not collisionFound:
            restartCounter += 1

            print("[INFO] No collisions found in the {} Message Space ...".format(messagesCount))
            print("-------------------------------------------------------\n")
            print("[INFO] Current Program Restart Count: ", restartCounter)
```


DEMO SCREENSHOTS:

Calling the startBirthdayAttack() initiates an attack on the BadHash44() function.

Starting the Birthday Attack on BadHash44()

```
In [12]:
# start birthday Attack
m1, m2, badHash = startBirthdayAttack()
```

```
[START] Starting Birthday Attack on BadHash44 Hash function ...

[INFO] Total Random Messages to be generated :  4194304
[INFO] Current Program Restart Count:  0
[INFO] Computing 4194304 256-bit Random Messages and respective 44-bit Hash value ...
[INFO] Writing Hash and Message pair to hash.txt file ...
[INFO] Searching for Collisions ...
[INFO] No collisions found in the 4194304 Message Space ...
-------------------------------------------------------

[INFO] Current Program Restart Count:  1
[INFO] Computing 4194304 256-bit Random Messages and respective 44-bit Hash value ...
[INFO] Writing Hash and Message pair to hash.txt file ...
[INFO] Searching for Collisions ...
[INFO] No collisions found in the 4194304 Message Space ...
-------------------------------------------------------

[INFO] Current Program Restart Count:  2
[INFO] Computing 4194304 256-bit Random Messages and respective 44-bit Hash value ...
[INFO] Writing Hash and Message pair to hash.txt file ...
[INFO] Searching for Collisions ...
[INFO] No collisions found in the 4194304 Message Space ...
-------------------------------------------------------

[INFO] Current Program Restart Count:  3
[INFO] Computing 4194304 256-bit Random Messages and respective 44-bit Hash value ...
[INFO] Writing Hash and Message pair to hash.txt file ...
[INFO] Searching for Collisions ...
[INFO] Collision Found!
[INFO] Collision on this Hash value:  40844a3555e
[INFO] Collision occured for these two input messages:
[INFO] M1:  5391c410e5acd5234ed85b743f95da462566dfa324ad44042aeca16e55e31789
[INFO] M2:  c326f315802b3c23b6c9ad75149f1c2c13dcb64336dc7ba38cd0a526296a6142
[END] Total Program Restarts:  3
```

For every iteration, it generates approx. 4 million 256-bit random messages and computes its 44-bit hash value. It stores all these hashes in a list, sorts them, and also writes them to a hash.txt file. It searches for a collision by checking if the adjacent values have the same hash. If a collision is found, the function returns the bad hash value and the two input messages for which the bad hash value was found. We also print the total number of program restarts required to get this output. If the collision is not found, the algorithm starts again from the message generation step and continues.

```
[INFO] Current Program Restart Count:  3
[INFO] Computing 4194304 256-bit Random Messages and respective 44-bit Hash value ...
[INFO] Writing Hash and Message pair to hash.txt file ...
[INFO] Searching for Collisions ...
[INFO] Collision Found!
[INFO] Collision on this Hash value:  40844a3555e
[INFO] Collision occured for these two input messages:
[INFO] M1:  5391c410e5acd5234ed85b743f95da462566dfa324ad44042aeca16e55e31789
[INFO] M2:  c326f315802b3c23b6c9ad75149f1c2c13dcb64336dc7ba38cd0a526296a6142
[END] Total Program Restarts:  3
```

As we can see, the output it returns is as follows:

Collision on this Hash value: 40844a3555e
M1: 5391c410e5acd5234ed85b743f95da462566dfa324ad44042aeca16e55e31789
M2: c326f315802b3c23b6c9ad75149f1c2c13dcb64336dc7ba38cd0a526296a6142

This output changes every time the attack is run as the input messages are random, and the chance of finding such a collision can be different every time. But as the messages count (4 Million) is a huge number, we will most likely find a collision within 3 iterations. We verify this input message by computing its hash using both SHA256 and BadHash44. Thus, we can see that the first 11 digits of the SHA256 output are the same. Hence BadHash44 is a bad choice for a hash function.

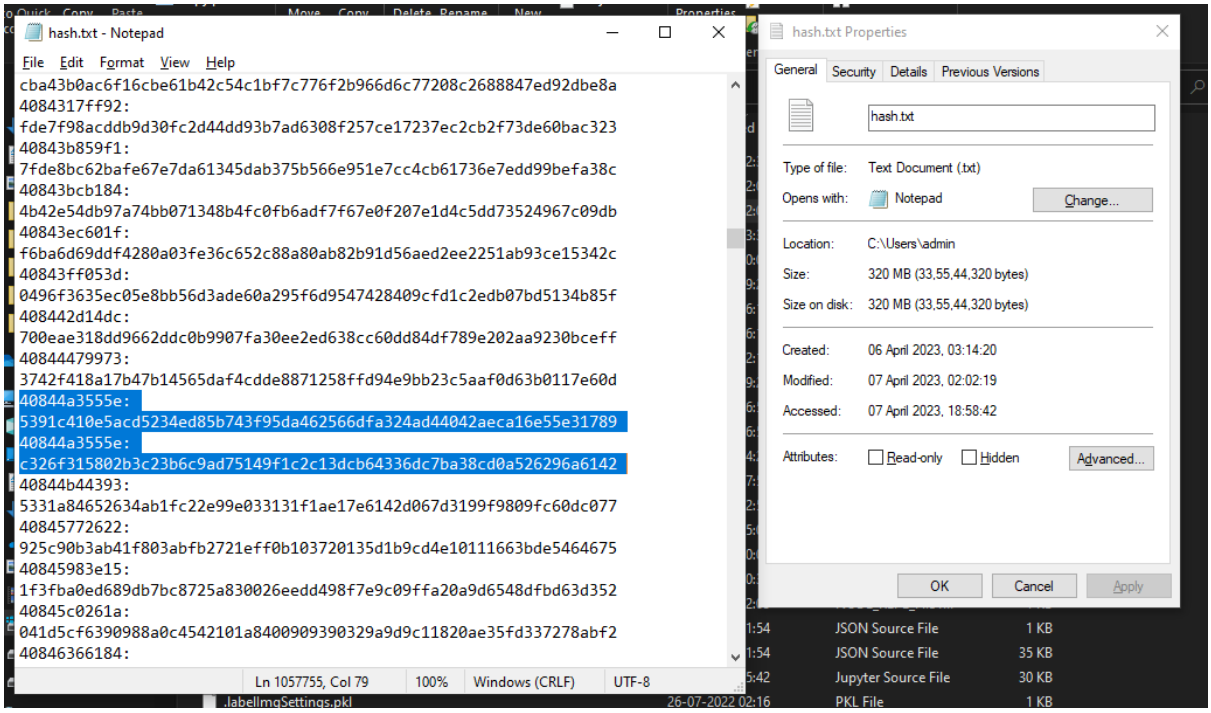
Verifying the Output Pair

In [13]:
```
print("For Message M1  : ", m1)
print("SHA256 Hash     : ", sha256(bytes.fromhex(m1)).hexdigest())
print("BadHash44 Output: ", BadHash44(bytes.fromhex(m1)))
print()
print("For Message M2  : ", m2)
print("SHA256 Hash     : ", sha256(bytes.fromhex(m2)).hexdigest())
print("BadHash44 Output: ", BadHash44(bytes.fromhex(m2)))
```

```
For Message M1  :  5391c410e5acd5234ed85b743f95da462566dfa324ad44042aeca16e55e31789
SHA256 Hash     :  40844a3555e1e0179bbd51fe8bb0f4ec08a6b1f4cda874cef77db5ad665a5c96
BadHash44 Output:  40844a3555e

For Message M2  :  c326f315802b3c23b6c9ad75149f1c2c13dcb64336dc7ba38cd0a526296a6142
SHA256 Hash     :  40844a3555ea1c1b5294f22860435c90a03d0bdb0e2d50ce3fb21d449a3e921a
BadHash44 Output:  40844a3555e
```

Hash.txt Output File:



As the output file is 320 MB, a small Excerpt of it is created which contains 100 (hash, message) pairs including the collision value.

References:

[1] Methods against DDOS Attacks, Mohammad Reza Khalifeh Soltanian, Iraj Sadegh Amiri, 2016 [https://www.sciencedirect.com/topics/computer-science/birthday-attack]

Submission Details:

The submission will include these

1. Project report .docx file
2. Code Files .zip file (This includes the .ipynb file and the hashExcerpt.txt file)