**PROJECT REPORT ON BIRTHDAY ATTACK**

**DESCRIPTION:**

**BIRTHDAY ATTACK:**

* It is a type of cryptographic attack which depends on the birthday paradox which tends to find a collision in the hash function. This birthday paradox is a statistical phenomenon that states that for example in a group of 50 people there is a slight chance that 2 people have birthdays on the same day. This attack takes advantage of this property by finding the 2 different messages which produce the 2 same hash values.
* This program is intended to implement the birthday attack on the BadHash44 algorithm, and it is constructed using the SHA-256 with only the first 44 bits to reduce the time of the attack. It can be implemented by taking many random messages to generate a larger table of hash values. So, if the list is large, we can able to find a collision in a hash values for a different messages.

**PROGRAM DESCRIPTION:**

* Here, we used python language to implement the birthday attack and at first, we need to install the crypto packages before compiling the codes by using the commands.

**!pip install pycryptodome**

**!pip install Crypto**

Text

Description automatically generated

* Since the program required for SHA 256 algorithm we used hashlib library and imported SHA 256 and also as need to Implement for generating random messages to find collisions so we used crypto library for importing get\_random\_bytes.
* This get\_random\_bytes will generate the random messages according to the input requirement. This will be later specified in the program as to what is the size that is required. Graphical user interface, text, application, chat or text message

  Description automatically generated
* Here, we defined a function call BadHash44 which implements us to generate 44 bit Badhash44 and returns hash\_string by using the hexdigest(). In the return value we need to find a hex bit which stores 4 stored bits which equal 4\*11 = 44 which is called as BadHash44.

Text

Description automatically generated

* This main program is used to find the collision and to add all the generated hashes to a .txt file or csv file. Here, for our convenience we are using hash.txt file. Here, this process is used to generate the random message set and generate the hashes for the same using the badhash function and then we will sort the hashes and then add them to the File. Here we have taken the value 1.2 \* 2^22 because in the description we have asked for 44 bits so according to the theorem the formula is 1.2 \* 2^n/2 = 1.2\*2^44/2 = 1.2\* 2^22.
* After the list is sorted, then we export them to a hash.txt file.

Text

Description automatically generated

Text

Description automatically generated

* Once the hash values (only the 44 bits data ie 11 characters in the hexadecimal format) and their corresponding messages are added to the hash.txt file after the sorting of them is performed, the collision is performed on them. This process is termed as an iteration.
* When once all the values are checked for the collision and there are no collisions occurred, then the whole process is repeated and the already present data in the hash.txt file is updated with the newly generated values and the search for the collision is performed. And if collision is not found, it is done again.
* This process is repeated until the collision is successfully found.
* Once the collision is found, the hash value to which the collision is obtained and the message to which the collision is associated is printed and the loop is terminated.

Graphical user interface, text

Description automatically generated

The birthday attack to find the collisions is written in the function named start\_attack(), and this is called for the process of execution of the program/birthday attack.

A picture containing chart

Description automatically generated

**EXPECTED/OBTAINED OUTPUT:**

* A hash.txt file is created for the random messages along with the first44 bit (11-character hexadecimal value).
* In the below screenshot, the collision results highlighted.
* The left 11 digit is the collision, and the corresponding is the text message associated with it.
* 
* After the program is started running for the attack, it could take 2 iterations to get the collision to be found. At first, we began performing the 1st iteration and there is no collision is not found for the 1st iteration. Then it started the 2nd iteration and now we could find the collision in 2nd iteration.
* Hence, there are two iterations, the reset required to achieve the collision in this case is 1.

Graphical user interface, text, application, email

Description automatically generated

The BadHash44 value is **1ceeef39d6f** and the random messages which we could find collisions are:

**b732dce359168a1c64a390c10a7d077153e22e4259943db57ce1bf070c236452** and **135245319a009c5e7431f11dd2cc058fb76f0a38dd8bfead76eb331a26fcf89f**

**REFERENCES:**

<https://docs.python.org/3/library/hashlib.html>

<https://blogboard.io/blog/knowledge/python-sorted-lambda/>

<https://www.w3schools.com/python/ref_file_writelines.asp>

<https://www.tutorialspoint.com/birthday-attack-in-cryptography>

Lecture 6-1, pg 34-38

<https://www.learnpythonwithrune.org/birthday-paradox-and-hash-function-collisions-by-example/>