Cryptography Session No.6

Summary 26-07-2022

- In Symmetric key, we had the issue that if someone gets my "key" then he can get back the original plain text, Which is very critical for us, But in **hashing**, if they even get my hash value, they don't have a way to get back the plain text from the "hash value".

- One of the Hash algorithms is "**Count**", Now whenever we covert the plain text to hash value using the "count" algorithm, Then "count" will just count the number of characters in our plain text and that would be the hash value for that data, for eg

Data	Hash Value
"Redhat" ->	6
"Sudhanshu" ->	9
"Accept" ->	6

Now from the hash value "6" No one can guess whether the original plain text would be "Redhat" or "Accept" " but this hash algorithm has a lot of bugs, one of the main issues in this algorithm is that it is not "Collision resistance".

- **Collision Resistance** - A best hashing algorithm should be collision-resistant, which Means No two "plain text" should give the same hash value.

- If two plain texts give the same hash value, Then anyone with the password "Accept" can unlock the account of the other user, whose original password was "RedHat", Why? Because both "Redhat" and "Accept" give the same hash value 6.

- **md5** is one of the well-known hashing algorithms, But in 2013 some attacks happened in this algorithm, So now we use md6, SHA-1, and SHA-2 hashing algorithms.

- **#Vim mytext.txt ->** "Redhat"
- **#openssl md5 my.txt ->** It will convert our data into a 128-bit hash value
- **#openssl sha1 my.txt ->** It will generate hash value with SHA1 algorithm
- Hashing and Encryption are two different things.
- In Linux when we create a Linux user, Linux uses an SHA-2 hashing algorithm to secure the password, We don't use it here for encryption, Because if someone gets my key then he would be able to decrypt the original passwords.
- **# cat /etc/shadow** -> At this location passwords of Linux users stored
- **Rainbow table** - By Hit and trial cracking a hash value to plain text is very hard, so the Hackers community has done, they have created their database where for a lot of "*characters combination*" they have stored their hash value, Now let's say Hackers got somehow access to any hash value stored in any server, Now they will search this hash value in their rainbow table if "hash value" got matched then they will get to know what was the original plain text of this hash value.

  Examples of rainbow tables –
  https://crackstation.net/
  https://md5decrypt.net/en/
- Now to be secure, that's why it's been advised to create a complex unique password because *general password combinations* are already present in the rainbow table
- **# cd /etc/pam.d ->** *Here linux maintain all authentication part in redhat linux*

- We use hashing not just for authentication but also for **Data integrity**, Let's say "A" has to transfer some data to "B" Then "A" will use some hashing algorithm to create its' hash value, "A" Will store this hash value with itself and will share the data to 'B' via n/wt, Now as "B" will receive the data, We will ask "B" to first convert data in hash value with the same algorithm If this hash value didn't matches with the hash value "A" have(which he created before sending data) then it means data was tempered somewhere in n/w by hackers. So here we are checking the **integrity** of data, If the hash value is the same then it means data is been not tempered, and "B" is receiving the actual data that was shared by "A". This is also called **Checksum** where we are checking the original signature with the signature we received So that we can verify we have received untempered data.