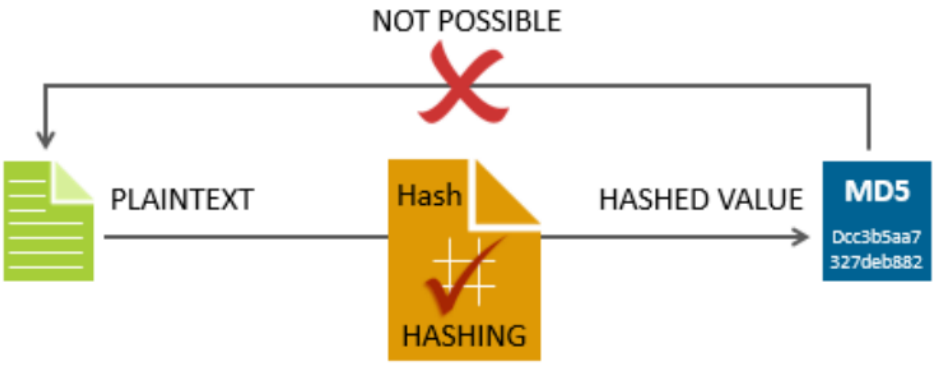
Hasing

Theory:

Hash function is a cryptographic function that maps data of arbitrary size to a fixed size value. The process of mapping data is called hashing and the output returned from the cryptographic function is called hash values or hashes.

Hash Function holds the following properties:

- Quick
- Irreversible
- Infeasible, making it impossible to find two different messages from the same hash value



Procedure:

Colab Notebook Link for this lab:
https://drive.google.com/file/d/1_fJVSPEnFYh-ZPnpw5_DJatilpLrW7tO/view?usp=sharing

1. Download password list: https://drive.google.com/file/d/1tqoWvRJu1h1uOfsxXiJ6G_ccY6DdsV_9/view?usp=share_link
2. Try to decrypt the password using a dictionary attack

RSA

Theory:

Public-key cryptography uses two separate keys, one for encryption (the public key) and one for decryption (the private key). Anyone with the public key can compute an encrypted message that only the owner of the private key can read.

RSA was the first algorithm that demonstrated this concept. Its security assumptions are based on complexity theory: computing the product of two prime numbers is easy (polynomial time), but there is no efficient algorithm for factoring them back (so far, all factorization methods are in the non-polynomial class).

The keys for the RSA algorithm are generated the following way:

1. Choose two different large random prime numbers ***p*** and ***q***
2. Calculate ***n = pq***
    - ***n*** is the modulus for the public key and the private keys
3. Calculate the totient: ***phi (n)=(p-1)(q-1)***.
4. Choose an integer ***e*** such that 1 < ***e*** < ***phi(n)***, and ***e*** is coprime to ***phi(n)***
    - ***e*** is released as the public key exponent
5. Compute ***d*** to satisfy the congruence relation ***d*e*** ≡ ***1 (mod(phi(n)))***
    - ***d*** is kept as the private key exponent

The public key is made of the modulus ***n*** and the public (or encryption) exponent ***e***.

The private key is made of the modulus ***n*** and the private (or decryption) exponent ***d*** which must be kept secret.

Encrypting a message: ***c = m ^ e (mod n)***

Decrypting a message: ***m = c ^ d (mod n)***

Small Exponential Attack:

In single-party RSA, we have some message p, public exponent e, and public modulus N, and $c \equiv p^e \mod N$.

If $p^e < N$, then we can solve $p^e = c$ easily and recover the message.

For example, if $e = 3$, $N = 187$, and $p = 2$, then $2^3 \mod 187 \equiv 8$.

The difficulty of breaking RSA comes from having $p^e > N$.

Suppose p is sent to several people. We'll have:

$$\begin{aligned} p^e &\equiv m_1 \mod N_1 \\ p^e &\equiv m_2 \mod N_2 \\ &\vdots \\ p^e &\equiv m_e \mod N_e \end{aligned}$$

The Chinese Remainder Problem algorithm allows us to solve this congruence mod $N_1 N_2 \cdots N_e$, and since $p < \min\{N_1, N_2, \ldots, N_e\}$, then $p^e < N_1 N_2 \cdots N_e$, and we can solve this congruence easily and recover p.

Procedure:

Colab Notebook Link for this lab:
https://drive.google.com/file/d/1_fJVSPEnFYh-ZPnpw5_DJatilpLrW7tO/view?usp=sharing

1. **Calculate** the congruent for d
2. Perform small exponent attack (cube root attack) on given data :

Suppose, Alice, Bob & Charli share an RSA system with e = 3 and public moduli 629, 2173, and 1159. Zachary communicates the values 529, 414, and 558 respectively.

Hint: https://www.johndcook.com/blog/2019/03/06/rsa-exponent-3/