Security & Cryptography

1. Motivation:
    a. understanding tools —> theory
    b. not a substitute for crypto courses
2. Entropy:
    a. measurement of randomness —> length of password
    b. safety depends on attacking models
    c. definition:

$$H = -\sum_{i=1}^{n} p_i \log p_i$$

3. Hash functions:
    a. fixed length of output
    b. SHA1 hash function —> used in Git

    `sha1sum` command:

    ```
    $ printf queen | sha1sum
    410114109270c8ffe4af1706adcad6e29c421f4d *-
    ```

    - determinstic: same input, same output
    - collision resistent
        - target collision resistent: given m_1, hard to find m_2, satisfying $h(m_1) = h(m_2)$
        - (stronger) collision resistent: hard to find $h(m_1) = h(m_2)$
    - hard to find the input

    c. applications:
        i. git log：

        ```
        $ git log
        commit e7dffdb6479d0f4da26e921a6357d974b06f8c57 (HEAD -> main, origin/main, origin/HEAD)
        ```

```
Author: Penrose819 <2420070554@qq.com>
Date:   Mon Dec 26 20:56:33 2022 +0800

    Readme

commit 8dee3ee7991153b02fce9f26ed59c105041bc0a4
Author: Penrose819 <96693507+Penrose819@users.noreply.github.com>
Date:   Mon Dec 26 20:50:29 2022 +0800

    Initial commit
```

ensuring no commit conflict

ii. short summary of a file

```
$ sha1sum README.md
a2964c8f3c2e72d06837a048e54d3e40d12aa688 *README.md
```

md5 code —> mirror source

iii. commit scheme:

given m, computing $h(m)$, check afterwards

4. Key derivation functions：
   a. slow to compute —> hard to brute force
   b. applications:
      i. producing keys from passpharse
      ii. storing log-in password (no plain-text)

      `salt = random(), value = KDF(password + salt)`
5. Symmetric cryptography：
   a. definition: symmetric —> **encryption and decryption use the same key**
      i. Gen —> k (key has high entropy)
      ii. $Enc_k(m)$ —> c
      iii. $Dec_k(c)$ —> m
   b. applications:

      untrusted cloud service —> avoid they use date to train their model, e.g. copilot

      passphrase —> key

      store `enc(file + key)`

6. Asymmetric cryptography
    a. asymmetric —> different key

        public key and private key

    b. definition:
        i. Gen —> k_{pub}, k_{pri}
        ii. $Enc_{k_{pub}}(m)$ —> c
        iii. $Dec_{k_{pri}}(c)$ —> m
    c. signing and verifying
        i. $Sign(m, k_{pub})$ —> s
        ii. $Veri(m, s, k_{pri})$ —> true / false
    d. applications:

        email encrption, private messaging

    e. key distribution:
        i. distributing the keys to real-world identities
        ii. web of trust, social proof
7. Case studies
    a. password manager:

        avoid password reuse,

    b. two-factor authentication:

        avoid leaking password

    c. SSH: (Secure Shell)

        `ssh-keygen` —> `public_key`, `private_key`

        `public_key` —> store —> server

        `private_key` —> encrypt on disk

        client —> challenge-response protocol —> log in