Cryptography I

What is crytography

confidentiality - symetric key encryption, public-key encryption

integrity: hash function, MAC, digital signature

data origin authentication: message originated source

entity authentication: know person is claimed one

non-repudiation

Symmetric encryption

Dec k(Enc k M) = M

hide: key + plaintext

Classic cipher

shift/ Ceasar cipher algorithm

(mod) - modulo

plaintext = D(key, ciphertext) = (ciphertext - key) mod 26

ciphertext = E(key, plaintext) = (plaintext + key) mod 26

ROT N(ROTATE)

https://www.dcode.fr/caesar-cipher

https://planetcalc.com/1434/

*security should depend only on the key

Brute force key search - frequency analysis

Vernam cipher

XOR operator

Encryption: c(i) = m(i) ⊕ k(i)

Decryption: M⊕K⊕K = M

Vernam cipher use any bit sourse as key

OTP (One-Time-Pad)

Vernam cipher with random key (not reused)

Encryption: C = P ⊕ K

length of plaintext and key are same

1. decide key
2. send message (calculation using +)
3. decrypt (calculation using -)

https://www.boxentriq.com/code-breaking/one-time-pad

K is not reused

Stream cipher

Block cipher: broken into fixed-length blocks before encryption, more modern, one block processed at a time

Stream cipher: block length is one bit/ char, require only limited buffering of data, letter by letter

Calculate Stream Cipher

Key Stream = Stream Cipher Algorithm (Key, Nonce)

Nonce - number used only once

C = P ⊕ KS

P = C ⊕ KS

Randomness for keys

true randomness from analog event is difficult to collect and too slow

Solution

- combine with deterministic algorithm with true randomness
- PRNG(seed) = random-looking string
- seed is unpredictable, out is unpredictable too
- seed = electrical noise in computer

Cryptographically Secure

1. next-bit test: unpredictable of the next bit with past bits
2. balanced: number of 1, 0 should be equal
3. non-linearity

Computational security

level computation required is far outweigh the computational resources of adversary

use case

stream cipher - RC4

key prefer random / derived from password

Entropy

source:

1. CPU support - Intel RdRand / backdoored;
2. Online Services(random.org, not trusted)
3. Cloudflare

Not good for cryptographic

```
# C/C++
srand(time(NULL));
rand();

# Java
Random randomGenerator = Random();
int randomInt = randomGenerator.nextInt(100);

# Python
random.randint(1,10)
```

Good entropy sources

```
# C++
# use API
CryptGenRandom();

# Java
SecureRandom random = new SecureRandom();
random.nextBytes(bytes);

# Python
os.urandom(n)
```