Secure Software Design

Andey Robins

Spring 23 - Week 5

Cryptography

Outline

- Cryptographic Primitives
- Hashing
- Symmetric Encryption
- Asymmetric Encryption
- KDAs
- Signing
- CAs and Certificates
- Applied Crypto

Serious Cryptography

A Practical Introduction to Modern Encryption

Jean-Philippe Aumasson

Foreword by Matthew D. Green

no starch press

Crypto Primitives

Crypto Primitives

- Hashes
- Entropy
- SRNGs
- Keys
- Ciphers

Hashes

Hashes are built upon the idea of one-way functions. Something easy to compute in one direction, but very difficult to reconstruct with just the hash.

$$\forall x, y; (\exists f; f(x) = y \iff \nexists f'; f'(y) = x)$$

In the above formula, the function `f` is the hashing function.

Entropy

Entropy is a measure of randomness. Formally, if your probability distribution is p_1, p_2, . . . , p_n, then entryopy is:

$$-p_1 * log(p_1) - p_2 * log(p_2) - ... - p_n * log(p_n)$$

Therefore, random 128 bit keys created over a uniform distribution have 128 bits of entropy:

$$2^{128} * (-2^{-128} * log(2^{-128})) = -log(2^{-128}) = 128 \text{ bits}$$

Key Takeaway: If you use a uniform distribution, you get as many bits as you expect.

RNGs

Cryptographically Secure RNGs vs Pseudo RNGs

I got a question teaching 1010 last semester: “Can you pick a random index into pi and use that to generate random numbers?”

RNGs

Cryptographically Secure RNGs vs Pseudo RNGs

I got a question teaching 1010 last semester: "Can you pick a random index into pi and use that to generate random numbers?"

It depends on the application. If we need cryptographically secure numbers, no because the next number is not independent from the previous since we could look it up and predict future numbers perfectly.

RNGs

Two parts:

1. A source of entropy
2. A crytpographic algorithm to produce random bits given a source of entropy

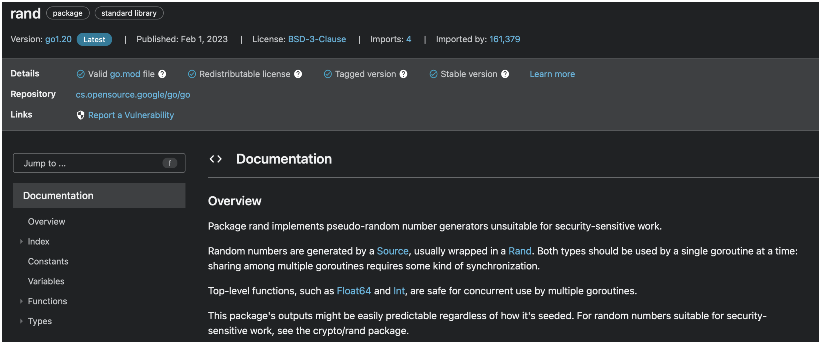

Figure 2: math/rand package for golang

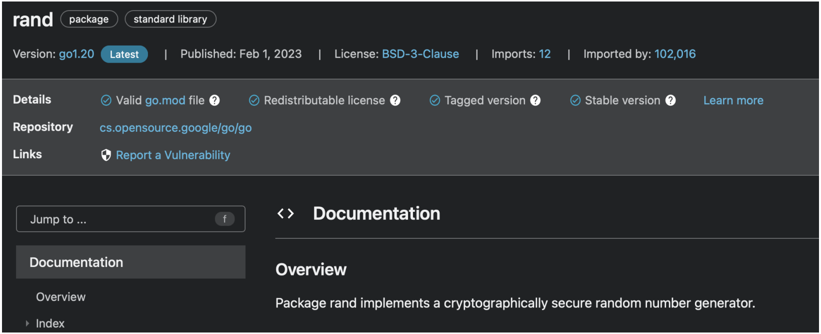

Figure 3: crypto/rand package for golang

Keys

Keys are an additional secret used in various crytpographic operations. The simplest form is seen in One Time Pad encryption, but it's probably better known in the sense of public and private keys.

One Time Pad

Given a Plaintext, P, and random key, K, we generate the ciphertext, C, with the following operation:

$$C = P \oplus K$$

Which decrypts due to the following rule:

$$C \oplus K = K \oplus K \oplus P = P$$

Ciphers

Ciphers, such as One Time Pad, are ways to *encrypt* information so that only those with a secret are able to read it. Some of the earliest are the Caesar cipher or Scytale cipher.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
t u v w x y z a b c d e f g h i j k l m n o p q r s
PLAIN TEXT
cipher text
Key = 19

Figure 5: A scytale cipher

Kerckhoff's Principle

Kerckhoff's Principle states that the secrecy of a cryptographic message should rely on the secrecy of the key and not the secrecy of the cipher.

Hashing

Collisions

The viability of a hashing algorithm relies on it being reistant to collisions. Collisions are two inputs which hash to the same value.

Collision Attacks

```
d131dd02c5e6eec4693d9a0698aff95c2fcab58712467eab4004583eb8fb7f89
55ad340609f4b30283e488832571415a085125e8f7cdc99fd91dbdf280373c5b
d8823e3156348f5bae6dacd436c919c6dd53e2b487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080a80d1ec69821bcb6a8839396f9652b6ff72a70
```

and

```
d131dd02c5e6eec4693d9a0698aff95c2fcab50712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325f1415a085125e8f7cdc99fd91dbd7280373c5b
d8823e3156348f5bae6dacd436c919c6dd53e23487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080280d1ec69821bcb6a8839396f965ab6ff72a70
```

Figure 6: Collision in MD5 with digest 79054025255fb1a26e4bc422aef54eb4

Example: Checksums

```
c396e956a9f52c418397867d1ea5c0cf1a99a49dcf648b086d2fb762330cc88d *ubuntu-22.04.1-desktop-amd64.iso
10f19c5b2b8d6db711582e0e27f5116296c34fe4b313ba45f9b201a5007056cb *ubuntu-22.04.1-live-server-amd64.iso
```

Figure 7: Ubuntu SHA256 digests for LTS version 22.04

Makes it easy to verify the authenticity of a file. When coupled with signatures, makes it very easy to verify the authenticity and ownership of very large files.

Password Hashes

Given our assumptions about hashes: they are irreversible and collision resistant, a very useful application becomes hashing passwords! Now, even if your password store is stolen, the user passwords aren't lost.

Example: Bcrypt

```typescript
export const getPassword = async (): Promise<string> => {
    return readPassIn("Password: ")
        .then((pass) => saltAndHash(pass));
};

const saltAndHash = (pass: string): string => {
    // 10 is the recommended default difficulty for bcrypt as of jan 2023
    const salt = bcrypt.genSaltSync(10);
    return bcrypt.hashSync(pass, salt);
};
```

Figure 8: An example of using bcrypt

Salt and Pepper

Salt and pepper are both pieces of information a principal can add to a password to secure it.

Salt - Added by the “backend,” definitely the much more common, completely best practice

Pepper - Added by the “user,” useful for redundancy or in low trust environments (i.e. putting the name of the website after the password in your password keeper.)

Symmetric Encryption

Asymmetric Encryption

Diffie-Helman Key Exchange

RSA

Elliptic Curve

NIST Curve Controvery

Key Derivation Algorithms

Signing

CAs and Certificates

Questions?

Next Time

This wraps up our fundamentals of cybersecurity.

- Security Design Patterns