PRNGs & DES

Luke Anderson

luke@lukeanderson.com.au

8th March 2019

University Of Sydney

Overview

1. Pseudo Random Number Generators

1.1 Sources of Entropy

1.2 Desirable PRNG Properties

1.3 Real PRNGs

- LCGs
- LFSRs
- RC4
- Other PRNGs

1.4 Using PRNGs

2. Stream Ciphers

2.1 Construction from PRNGs

3. Data Encryption Standard

3.1 Introduction

3.2 History

3.3 NSA's involvement

3.4 Feistel Networks

3.5 DES Internals

3.6 Diffusion and Confusion

Pseudo Random Number Generators

Pseudorandom Number Generators

A source of random numbers are essential in many occasions:

- Session Keys
- Shuffling of Cards
- Challenges
- Nonces

Computers are inherently deterministic. As a result, true randomness is a difficult thing to come by.

Since we can't get randomness easily, we use **pseudorandom number generator** functions (PRNGs) to generate what appears to be statistically random output.

A **cryptographically secure pseudo random number generator** (CSPRNG) is a type of PRNG whose properties make it suitable for use in cryptography.

Sourcing Randomness

PRNG functions produce the same sequence of seemingly random output when provided with particular "**seed**" data.

Since a computer is deterministic it must extract randomness (entropy) from an external, truly random source. This could be something like:

- Thermal noise of hard drives
- Low-order bit fluctuations of voltage readings
- User input
- Geiger counter click timing

Randomness can actually be *really* hard to come by:

U.S. Patent 5,732,138

Method for seeding a pseudo-random number generator with a cryptographic hash of a digitization of a chaotic system.

"**Lavarand**" by Silicon Graphics

Properties of PRNGs

Desirable properties of PRNGs include:

- Repeatability
- Statistical randomness
- Long period / cycle
- Insensitive to seeds

PRNGs are often broken by:

- Statistical tests that find patterns or biases in the output sequence
- Inferring the state of the internal registers from the output sequence

PRNGs are usually **critically important parts of a cryptosystem**.
They are often a single point of failure.

e.g. Android Bitcoin wallet apps vulnerable to theft

Linear Congruential Generators

An LCG generates a sequence $x_1, x_2, \dots$ by starting with a *seed* x_0 and using the rule:

$$x_{n+1} = (ax_n + b) \bmod c$$

where a, b, and c are fixed constants.

- The *period* of the PRNG is at most c.
- **Must not** be used for security purposes – it's easily predictable.
- Only two values x_i, x_{i+1} are needed to determine a and b.
- Commonly found in libraries, e.g. the Unix `rand()` function.

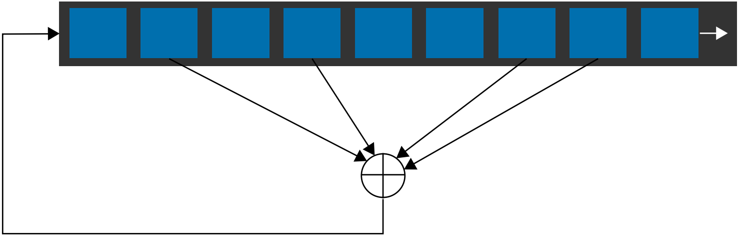
Don't Use LCGs Where Security Matters

An LCG was once used by an online casino who were so sure of their code that they published their algorithms...

...the results were as one would expect.

Linear Feedback Shift Registers

A Linear Feedback Shift Register (LFSR) simply combines the bits of a series of registers, and shifts the output onto the register.



- The *seed* is the initial value of the register.
- Easy and fast in hardware (1 bit per clock).
- **Problem:** tap configuration can be determined from $2n$ output bits, where n is the length of the LFSR period.

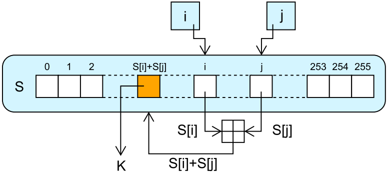
RC4 Stream Cipher

RC4 is a stream cipher which has wide applications in cryptography.

- At one point, RC4 was used to encrypt $>$ 50% of all SSL traffic.
- It is the core algorithm of Wired Equivalent Privacy (WEP)

Based on permutations of a 256 byte array, the seed is the initial array value. RC4's key scheduling algorithm has known problems (e.g. WEP weakness)

```
i := 0
j := 0
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap values of S[i] and S[j]
    K := S[(S[i] + S[j]) mod 256]
    output K
endwhile
```



Other PRNGs

ANSI X9.17
Based on 3DES

DSA PRNG
Based on SHA or DES

RSAREF PRNG
Based on MD5 hashing and addition modulo 2^{128}

Tips for using PRNGs

- Be extremely careful with PRNG seeds!
- Hash PRNG inputs with a timestamp or counter
- Reseed the PRNG occasionally
- Use a hash function to protect PRNG outputs if PRNG is suspect

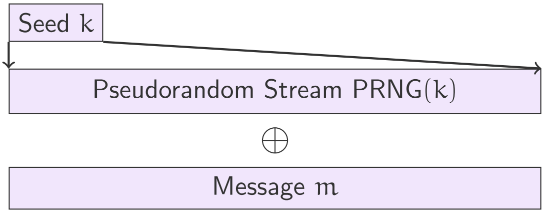
Stream Ciphers

Stream Ciphers from PRNGs

In a one-time-pad, we have a perfectly random r the same size as the message m, and the ciphertext is $c = r \oplus m$.

- **Idea:** Replace r with a pseudorandom stream.
- The seed for the PRNG is the key k.
- Encryption: $E_k(m) = m \oplus \text{PRNG}(k)$.
- Decryption: $D_k(c) = c \oplus \text{PRNG}(k)$.

Whenever a key k is expanded to a large pseudorandom stream, this is called a *stream cipher*.



Stream Ciphers

Advantages of using a stream cipher:

- Ease of implementation and use.
- Secure PRNGs can be a lot faster than block ciphers.

The security of a stream cipher directly depends on the security of the pseudorandom number generator.

- It must be computationally hard to find the seed k, or the sequence $\mathrm{PRNG}(k)$.
- The seed k must be used *only once*.
- The PRNG period must be at least as long as the message.

As with the one-time-pad, stream ciphers by themselves only ensure *secrecy*. The message may still be modified in transit.

Data Encryption Standard

Data Encryption Standard (DES)

Block Ciphers

A *block cipher* is a pair of encryption/decryption algorithms (E, D) operating on *blocks* of a fixed length B. Both algorithms take a K-bit key k, and for any block b:

$$D_k(E_k(b)) = b$$

DES is a block cipher operating on 64-bit blocks, using a 56-bit key.

- Developed in the early 1970's at IBM.
- "Tweaked" by the NSA (National Security Agency) before release in 1977.
- The world's most heavily analysed and used cipher.

A brief history of DES

1970s	IBM Research team led by Feistel develops the LUCIFER cipher (128-bit blocks and keys)
1973	NBS (now NIST) asks for a proposed data encryption standard
1974	IBM develops DES from LUCIFER
1975	NSA “fixes” DES: shortens key from 64 to 56 bits, and modifies some S-boxes (substitution boxes)
1977	DES adopted as a standard.
1991	Biham and Shamir discover *differential cryptanalysis*, apply their new technique to DES, and find that the NSA’s modifications had improved security
1993	Michael Wiener of Nortel theorises a USD$1M machine could crack DES in 3.5 hours using general purpose hardware
1997	DES cracked by brute force by `distributed.net` in 96 days NIST asks for a proposal for AES (Advanced Encryption Standard)
1999	DES cracked in 24 hours by `distributed.net` and the EFF USD$250,000 Deep Crack machine
2000	Rijndael accepted as AES (128/192/256-bit keyspace, 128-bit blocks)

The NSA’s involvement in DES

The NSA’s modifications to DES were thought to be adding a “back door”.

- 20 years later (1990s), academia independently rediscovered *differential cryptanalysis* (DC).
- DC had been discovered by IBM in the 1970s (and used in the construction of DES), but IBM were gagged by the NSA.
- The NSA had used DC to *strengthen* DES, while no-one else was aware it existed.

> “NSA doesn’t want a strong cryptosystem as a national standard, because it is afraid of not being able to read the messages. On the other hand, if the NSA endorses a weak cryptographic system and is discovered, it will get a terrible black eye.” — EFF, 1998

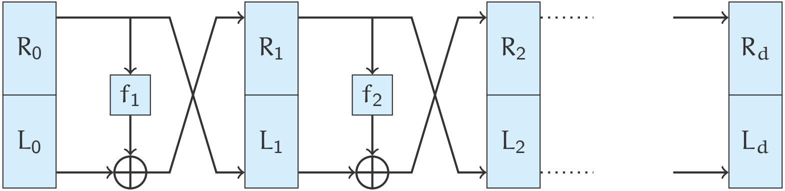
Feistel Networks

Recall: a block cipher $E_k : \{0,1\}^n \to \{0,1\}^n$ must be *invertible*.

- ○ Hard: coming up with cryptographically secure invertible functions.
- ○ Easier: coming up with pseudorandom functions (such as hashes).

Feistel Network

Given d pseudorandom functions $f_1, \ldots, f_d$, where each f_i maps n bits to n bits, a *Feistel Network* combines these functions into a secure invertible function F, mapping $2n$ bits to $2n$ bits.



Feistel Networks

Algebraically, we have d functions $f_1, \ldots, f_d$, and split our initial input into two: (L_0, R_0). The Feistel network is then:

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= f_i(R_{i-1}) \oplus L_{i-1} \end{aligned}$$

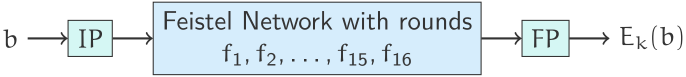
Finishing with the final value (L_d, R_d).

- The whole network is invertible because each step is invertible.
- In fact, the inverse network is identical, but with the function order reversed: $f_d, f_{d-1}, \ldots, f_2, f_1$.
- When used in a cipher, the functions $f_1, \ldots, f_d$ are called *round functions*.

DES Internals

DES takes a 56-bit key k and a 64-bit block b to be encrypted.

- An *initial permutation* IP is applied to the block.
- The block is then fed into a 16-round Feistel network, with round functions $f_i(x) = F(x, k_i)$, where $k_1, \ldots, k_{16}$ is the *key schedule* derived from k, and F is the DES *round function*.
- A *final permutation* FP is then applied to the output.



b

IP

Feistel Network with rounds $f_1, f_2, \ldots, f_{15}, f_{16}$

FP

$E_k(b)$

The overall structure of DES

The round function $F(x, k_i)$ is the core of DES.

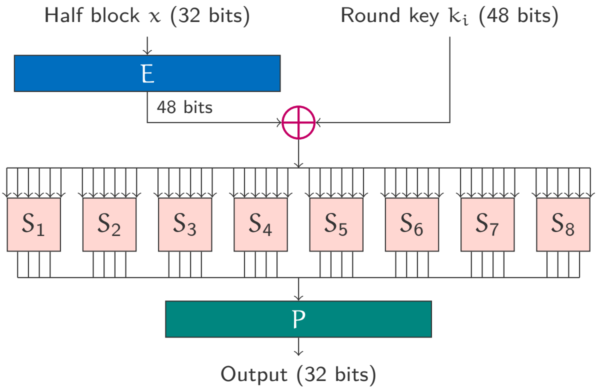
The DES Round Function

The round function $f_i(x) = F(x, k_i)$ consists of:

○ E, an *expansion permutation* widening x from 32 to 48 bits.

○ S_j, the *substitition boxes* (S-boxes) which collapse 6 bits to 4 bits.

○ P, a fixed *permutation* (P-box).



Diffusion and Confusion

Diffusion is the dissipation of statistical information present in the plaintext.

- Flipping a bit in the plaintext should result in half the bits of the ciphertext changing.
- Flipping a bit in the ciphertext should result in half the bits of the plaintext changing.

Confusion is making the relationship between the key and the ciphertext as complicated as possible.

- Each bit of the ciphertext should depend on multiple bits of the key.
- Even if an attacker gathers many (plaintext, ciphertext) pairs encrypted under the same key, they should not be able to deduce the key.

Diffusion and Confusion with S and P-boxes

When highly nonlinear S-boxes are combined with good P-boxes, both the properties of confusion and diffusion arise.

- Having linear S-boxes would make the whole of DES a linear function.
- Having P-boxes not spreading bits around enough would allow DES to be broken down into smaller independent subproblems.

Qualitatively, “good” S and P-boxes “work together”:

- S-boxes will be highly nonlinear, and flipping an input bit should result in half the output bits flipping.
- The P-box following this up should distribute those bits evenly across S-boxes in the next round.
- The P-box must diffuse the round key evenly over the whole block.