Part I

Stream Ciphers

1. Stream Ciphers

Making OTP practical

Instead of using a random key of n bits, use a pseudorandom key of n bits obtained by expanding a seed of s bits.

Definition (Pseudorandom Generators, PRG)

A PRG is a function:

$$G\colon \{0,1\}^s \to \{0,1\}^n \text{ with } n \gg s$$

G must be efficiently computable by a deterministic algorithm

Notation: $G(k)[i]$ is the $(i-1)$th bit of the stream with seed k. We use 0-based indexing.

Stream Cipher

Given $\mathcal{M} = \{0,1\}^n$ and $\mathcal{K} = \{0,1\}^s$ with $n > s$.

Definition (Stream Cipher)

$\mathsf{Gen}()$	k uniformly at random from $\mathcal{K}$
$\mathsf{Enc}(k, m)$	$c[i] = G(k)[i] \oplus m[i] \quad 0 \le i < n$
$\mathsf{Dec}(k, c)$	$m[i] = G(k)[i] \oplus c[i] \quad 0 \le i < n$

- Since $|\mathcal{K}| < |\mathcal{M}|$, then a stream cipher cannot have perfect secrecy.
- We need a different definition of security.
- Security (or lack of) will depend on the properties of the specific G used.

Properties of PRGs

A secure PRG must be unpredictable.

Definition (Predictability)

A PRG $G(k)$ is predictable if there exists an efficient algorithm $\mathcal{A}$ and an index i such that, given the first i bits of the stream, it outputs a correct guess for the next bit with probability larger than 1/2 by a non-negligible quantity.

$$\Pr\left\{\mathcal{A}\Big(G(k)[0],\ldots,G(k)[i-1]\Big) = G(k)[i]\right\} > \frac{1}{2} + \varepsilon$$

Predictability

Remember that $G(k)[i] = c[i] \oplus m[i]$, so it is not difficult to obtain. Then, an attacker knowing the first i bits of the plaintext can find the following bit of the keystream. (Not necessarily the key k, but do we care?)
A PRG is unpredictable if such efficient algorithm does not exist.

New Security Assumptions

Information-Theoretic Security

- Computationally unbounded adversaries.
- No information is leaked.

Computational Security

- Security is guaranteed against efficient adversaries. Computational resources are bounded.
- The adversary can succeed with negligible probability.

Concrete Model for Computational Security

Concrete Model

The efficient Algorithm $\mathcal{A}$:

- runs for at most a given time t
- guesses with probability better than $1/2$ by at most a given constant ε

The constants depends on the application, the attack scenario, the amount of data, how long we want the crypto to be secure, etc.
Typical values: $t = 2^{80}$ CPU cycles and $\varepsilon = 2^{-60}$.

Asymptotic Model for Computational Security

Security depends on a *security parameter* λ, which governs the encryption parameters. Generally, the security parameter is the key size.

Asymptotic Model

The efficient Algorithm $\mathcal{A}$:

- runs in polynomial time in λ.
- guesses with probability better than $1/2$ by at most a given negligible function $\varepsilon(\lambda)$

$\varepsilon(\lambda)$ is negligible if it goes to zero faster than any polynomial in λ^{-1}.

Examples of predictable PRGs

The linear congruential generator is a popular PRG for usage in simulation software.

```
procedure LINEAR CONGRUENTIAL GENERATOR($k$)
    $r[0] = k$
    for $i = 1$ to $n - 1$ do
        $r[i] = ar[i-1] + b \bmod p$    ▷ $p$ is a prime number
        output $r[i]$
    end for
end procedure
```

Examples of predictable PRGs

It is easy to build a predictor for the LCG.

```
function $\mathcal{A}(x)$
    return $ax + b \bmod p$
end function
```

Given $r[0]$, $\mathcal{A}(r[0])$ predicts $r[1]$ with success probability 1.

Examples of predictable PRGs

The standard random generator of the linux C library.

```
procedure RANDOM(k)
    $r[0] = k$
    for $i = 1$ to $n - 1$ do
        $r[i] = r[i-3] + r[i-31] \bmod 2^{32}$
        output $r[i]/2$                    ▷ interger division
    end for
end procedure
```

Examples of predictable PRGs

The division drops some bits, but there is enough material to make a good guess.

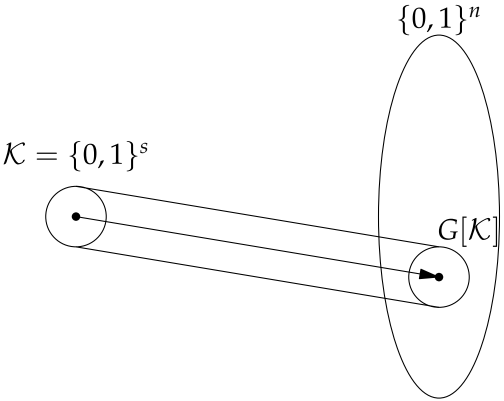
```
function $\mathcal{A}(x_0, \dots, x_{30})$
    return $2x_{28} + 2x_0 \bmod 2^{32}$
end function
```

$\mathcal{A}(r[0], \dots, r[30])$ predicts $r[31]$ with probability 1 except for the last two bits.

1. Stream Ciphers

Distinguishability

Consider the *image* of G, $G[\mathcal{K}]$. Then $G[\mathcal{K}]$ is a very small subset of $\{0,1\}^n$.



Distinguishability

We can label any string in $G[\mathcal{K}]$ as *not random* and any other string of n bits as *random*.

Consider an n-bit string x. Is x random?

If we could enumerate all the elements of $G[\mathcal{K}]$, it would be easy to test if x is random or not. But this has exponential complexity in s and is not feasible.

We say that G is distinguishable from a true random generator if there exists some efficient algorithm that can guess if any x is random with non-negligible success.

If such algorithm does not exist, then G is indistinguishable from a true n-bit random generator.

Advantage

Definition (Advantage)

Let $G(k)$ a PRG and $\mathcal{A}(x)$ a statistical test giving 0 if x is not random and 1 if x is random. Then

$$\text{Adv} = \Big| \Pr[\mathcal{A}(G(k)) = 1] - \Pr[\mathcal{A}(r) = 1] \Big|$$

for any random key k and random string r.

The advantage ranges from 0 to 1. Adv larger than 0 means that $\mathcal{A}$ is a good distinguisher. A negligible Adv means that $\mathcal{A}$ cannot distinguish.

Example

A test giving always 0 has an advantage equal to zero.

$$\mathcal{A}(x) = 0 \implies \text{Adv} = 0$$

Advantage (cont.)

Example

Suppose $G(k)$ satisfies $G(k)[0] = 1$ for $2/3$ of the keys.
Now consider the following $\mathcal{A}$.

```
procedure $\mathcal{A}(x)$
    if $x[0] = 1$ then
        output 0                                   ▷ Not random
    else
        output 1                                   ▷ Random
    end if
end procedure
```

$$\mathrm{Adv} = |\Pr[G(k)[0] = 0] - \Pr[r[0] = 0]| = \left|\frac{1}{3} - \frac{1}{2}\right| = \frac{1}{6}$$

Cryptographically Secure PRG

Definition (Secure PRG)

G is a secure PRG if for all efficiently computable tests $\mathcal{A}$, then Adv is negligible.

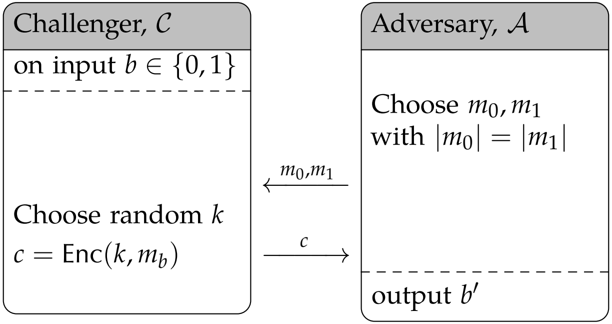
No provable secure PRG exist, but some heuristic candidates.

Theorem

G is a secure PRG if and only if G is an unpredicatable PRG

Semantic Security

Consider the experiment $b' = \mathsf{Exp}(b)$.


Challenger, $\mathcal{C}$
on input $b \in \{0,1\}$
Choose random k
$c = \mathsf{Enc}(k, m_b)$
Adversary, $\mathcal{A}$
Choose m_0, m_1 with $|m_0| = |m_1|$
m_0, m_1
c
output b'

Let $\mathsf{Adv}[\mathcal{A}, \mathcal{E}] = |\Pr[\mathsf{Exp}(0) = 1] - \Pr[\mathsf{Exp}(1) = 1]|$.

Definition (Semantic Security Experiment)

The cipher $\mathcal{E}$ is *semantically secure* if $\mathsf{Adv}[\mathcal{A}, \mathcal{E}]$ is negligible for any efficient $\mathcal{A}$.

Semantic Security and Stream Ciphers

Theorem

Any information-theoretic secure cipher is semantically secure.

Theorem

If $G(k)$ is a secure PRG then the stream cipher derived from G is semantically secure.

Intuition: if $G(k)$ is indistinguishable from a true random key, then a stream cipher is indistinguishable from OTP.

1. Stream Ciphers

Real-world Stream Ciphers

Several Stream Ciphers have been used in the past and currently in use. Many of them are broken.

Pro:

- very fast also in software implementations
- small footprint in hardware and software

Con:

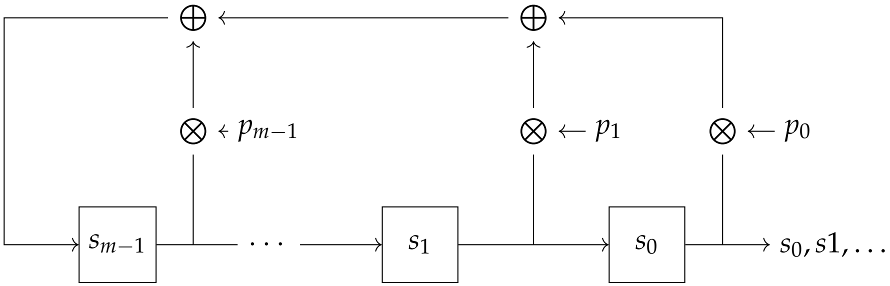
- security properties less understood

Using Nonces

Modern communications are packet-oriented or message oriented, meaning that many messages must use the same key. Using a very long stream over multiple messages is impractical because it requires a re-synchronization in case packets are reordered or lost.
Modern Ciphers expand a key k and a *nonce* r into a keystream.
The nonce is a throwaway random number used only once for any given key. Can be included in the message with no information loss.
The Encryption algorithm becomes:

$$\mathsf{Enc}(k, m; r) = m \oplus G(k; r)$$

Linear Feedback Shift Registers



- Historically used as PRG, very efficient in hardware, good statistical properties.
- Predictable and thus very bad for cryptography. Popular in the past because very fast (DVD, GSM A5), but the state of the generator is very easy to recover.

Linear Feedback Shift Registers

The initial state of the LFSR is the key k or a nonce concatenated to the key. In general, the output can be described as:

$$G(r\|k)[i+m] = \sum_{j=0}^{m-1} p_j s_{i+j} \bmod 2$$

with initial values $G(r\|k)[0..m-1] = r\|k$.
Since the state of the generator has m bits, the maximum sequence length is $2^m - 1$. The specific period depends on the p_j and can be much shorter. It is trivial that every output sequence longer than the period is predictable.
Note that the LFSR is predicatable even for sequences shorter than the period. (Linearity and security do not go well together)

Nonlinear Feedback Shift Registers

Combine multiple LFSR with nonlinear blocks. One such example is Trivium. It has resisted cryptanayisis attempts, but the security margin is now slim.

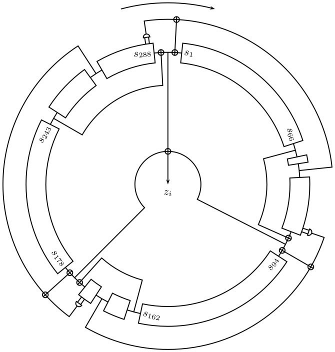


Figure: The Trivium cipher

Security Margin

Generally an cryptanalysis attack might fail against the full cipher, but may succeed against a simplified version of the cipher. The security margin is a measure of how simpler is the cipher that was broken. For example, Trivium specifies 1152 initialisation rounds and the broken version has 799 rounds, so the margin is 353 rounds. Whether this is large or small is subjective.

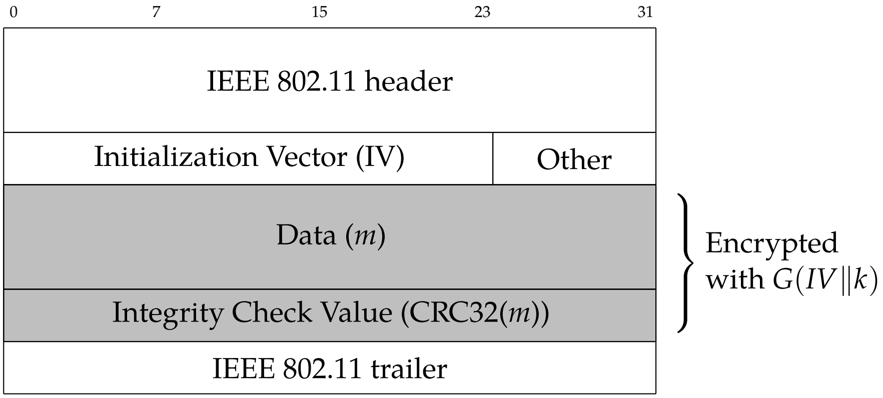
RC4 (Rivest, 1984)

Very popular but now not secure.

- Key size is 40 – 2048 bit.
- Biases in the initial output make it predictable.
- Various other related key attacks.
- No support for nonces. Nonces are concatenated to the key and shorten the key size.
- Should not be used, but still widely used.

IEEE 802.11b WEP

Stations (STA) and AP send frames protected with the RC4 cipher.



0
7
15
23
31
IEEE 802.11 header
Initialization Vector (IV)
Other
Data (m)
Integrity Check Value (CRC32(m))
IEEE 802.11 trailer
Encrypted with $G(IV\|k)$

IV repeated every $2^{24} \approx 16M$ frames.
Some devices reset IV to 0 at boot.

Salsa20/20 and ChaCha20 (Daniel J. Bernstein, 2007)

Salsa20 is a family of stream ciphers with different numbers of rounds and cost/security tradeoffs.
Salsa20/20 and ChaCha20 (with 20 rounds) are the recommended versions for general use. ChaCha20 is available for use with TLS.

- Key size is 128 or 256 bits plus a 64-bit nonce.
- Each 512-bit block of the keystream can be accessed randomly using a 64-bit block number
- Max output size is 2^{73} bits
- 4–14 cycles per byte on x86
- Not provably secure, but best simplified version broken has 7 rounds.

Practical RG/PRG

PRG are used also outside of stream ciphers (e.g. for generating keys, nonces and IVs)

Hardware Random Number Generators use physical processes such as noise in semiconductors. Might be slow, but provide a constant rate of random numbers.

Blocking Random Sources use the timing of hardware interrupts (e.g. keyboard, network frames arrivals) as a source of randomness. They block if there are not many interrupts. e.g. Linux `/dev/random`

NonBlocking Random Sources use a blocking random source to feed a PRG to avoid blocking if randomess accumulates too slowly. e.g. Linux `/dev/urandom`