1 Stream Ciphers

1.1 Information theoretic security and the one time pad

1.1.1 Symmetric Ciphers: definition

A cipher defined over $(K,\, M,\, C)$ is a pair of "efficient" algorithms $(E,\, D)$ where

$$E : K \times M \to C$$
$$D : K \times C \to M$$

K - key space
M - message space
C - cipher text
E - encryption algorithm
D - decryption algorithm
The requirement is that the algorithms are consistent (satisfy correctness property)

$$\forall m \in M,\, k \in K : \mathbf{D(k,\, E(k,\, m)) = m}$$

Bolded is the consistency equation, which all ciphers must satisfy.
Efficient means that a cipher runs in polynomial time, concrete time constraints.
E is oftne randomized
D is always deterministic

1.1.2 The One Time Pad

First example of a "secure" cipher.

$$M = C = \{0,\, 1\}^n$$

set of all n bit binary strings.

$$K = \{0,\, 1\}^n$$

Key is a random bit string as long as the message.

$$C := E(k,\, m) = k \oplus m$$

$$D(k,\, c) = k \oplus c$$

Indeed:

$$\begin{aligned} D(k,\, E(k,\, m)) &= D(k,\, k \oplus m) \\ &= k \oplus (k \oplus m) \\ &= (k \oplus k) \oplus m \\ &= 0 \oplus m \\ &= m \end{aligned}$$

Given a message, m, and it's One Time Pad (OTP) encryption, c, the key is computed:

$$k = m \oplus c$$

OTP is very fast in enc/dec, but has long keys and so hard to use in practice.
Is the OTP a good cipher?
First we have to answer what is a good cipher.

1.1.3 Information Theoretic Security

(Shannon 1949)

Basic idea: Cipher text (CT) should revel no "information" about plain text (PT)

Def: A cipher (E, D) over (K, M, C) has perfect secrecy if:

$$\begin{aligned} \forall m_0,\, m_1 \in M & & |m_0| = |m_1|\, \forall c \in C \\ Pr[E(k,\, m_0) = c] &= & \mathbf{Pr}[\mathbf{E}(\mathbf{k},\, \mathbf{m}) = \mathbf{c}] \\ & & \text{where}(\text{k} \overset{R}{\leftarrow} \text{K}) \\ & & k\, is\, uniform\, in\, K \\ & & \text{choose random key} \\ & & \text{from the set K} \end{aligned}$$

This means that if I am an attacker and I intercept a particular cipher, c, then the probability that the cipher text is the encryption of m_0 is the same as the probability that it's the encryption of m_1. So that means that if we're encrypting messages with the OTP, the most powerful adversary can learn nothing about the plain text from the cipher text. There's no CT only attack on the cipher that has perfect secrecy. Other attacks may be possible.

Lemma: OTP has perfect secrecy

Proof:

$\forall m,\, c:\ \underset{K}{PR}[E(k,\, m) = c] = \frac{\#keys\, k \in K}{|K|}$

Probability of random choice of key = number of keys in K/total number of keys.

Suppose we have a cipher such that

$\forall m,\, c:\ \#\{k \in K:\ E(k,\, m) = c\} = const.$

the cipher has perfect secrecy.

Let $m \in M$ and $c \in C$. One OTP key maps m to c.

Proof:

For OTP: if $E(k,\, m) = c$ then

$$\begin{aligned} k \oplus m &= c \\ &\Rightarrow k \oplus m = c \\ &\Rightarrow k = m \oplus c \\ &\Rightarrow \#\{k \in K:\ E(k,\, m) = c\} = 1 \qquad \forall m,\, c \\ &\Rightarrow \text{OTP has perfect secrecy} \\ &\Rightarrow \text{OTP no CT only attack, but others possible} \end{aligned}$$

The bad news lemma...

Theorem: perfect secrecy $\Rightarrow |K| \geq |M|$, the length of key in the cipher must be at least the length of message

Hard to use in practice.

1.2 Stream ciphers and pseudo random generators

1.2.1 Stream Ciphers: making OTP practical

idea: replace “random” key by “pseudorandom”

PRG: Pseudo Random Generator is a function, G, that takes a seed and maps it to a much larger string

$$G : \{0,\, 1\}^S \to \{0,\, 1\}^n,\ n \gg S$$

Function G is efficiently computable by a deterministic algorithm. Output should look random.

To use this to build a stream cipher, we’re going to use the seed as our key and then use the generator to expand the seed to a random looking sequence and XOR it with the message. That will give us our cipher text.

$$\begin{aligned} c = E(k,\, m) &:= m \oplus G(k) \\ D(k,\, c) &:= c \oplus G(k) \end{aligned}$$

Can a stream cipher have perfect secrecy? No, the key is shorter than the message.

- Need a different definition of security
- Security will depend on specific PRG

1.2.2 PRG must be unpredictable

Suppose PRG is predictable, that meanst that there exists some i such that if i give you the first i bits of the output, there’s an efficient algorithm that will compute the rest of the stream.

$$\exists i :\ G(k)|_{1,\ldots,i} \overset{algo}{\to} G(k)|_{i+1,\ldots,n}$$

If this is the case, then the stream cipher would not be secure. Suppose an attacker intercepts a cipher, then we have a problem because suppose by some prior knowledge, attacker knows the initial part of the message starts with some known value. The attacker could XOR the cipher text with the prefix and that would give him the prefix of the pseudo random sequence. Then he could predict the rest of the message.

It turns out that even predicting even 1 bit of the output, given the first i bits, predicting the $i_{th} + 1$ bit $G(k)|_{1,\ldots,i} \to G(k)|_{i+1}$, is problematic.

We say that $G : K \to \{0,\, 1\}^n$ is **predictable** if there exists an efficient algorithm, A, and there is some position, i, between 1 and -1, such that if we look at the probability over a random key, if we give this algorithm the prefix of the output, the probability that it can predict the next bit of the output is $\geq \frac{1}{2} + \varepsilon$ for some non-negligible ε ($\varepsilon \geq 1/2^{30}$):

$$\underset{k \overset{R}{\leftarrow} K}{PR} \left[A|G(k))|_{1,\ldots,i} = G(k)|_{i+1}\right] \geq \frac{1}{2} + \varepsilon$$

<u>Def</u>: PRG is unpredictable if it is not **predictable** $\Rightarrow \forall i$: no “efficient” adversary can predict bit $(i + 1)$ for “non-negligible” ε.

1.2.3 Weak PRGs (do not use for crypt)

Linear congruential generator. It has three parameters: a, b, p where a and b are integers and p is a prime. The generator is defined as follows:

$$r[0] \equiv seed$$

the way you generate randomness is run iteratively through the following steps:

$$r[i] \leftarrow a \times r[i-1] + b \mod p$$

that outputs a few bits of $r[i]$, then $i++$ and repeat. This generator has some statistical properties (#0s≈#1s) that make it easy to predict.

Another example is random generator implemented in glibc.

1.2.4 Negligible and non-negligible

In practice: ε is a scalar and

εnon-neg:$\varepsilon \geq \frac{1}{2^{30}}$(likely to happen over 1GB of data)

εnegligible:$\varepsilon \geq \frac{1}{2^{80}}$(won't happen over life of key)

If you happen to use a key for encrypting a gigabyte of data, then an event with a probability of happening of $\frac{1}{2^{30}}$ will likely happen after about a gigabyte of data. Therefore $\frac{1}{2^{30}}$ is non-negilible.

In theory: ε is a function $\varepsilon : \mathbb{Z}^{\geq 0} \rightarrow \mathbb{R}^{\geq 0}$ and

εnon-neg:$\exists d : \varepsilon(\lambda) \geq \frac{1}{\lambda^d}$ inf. often $(\varepsilon \geq \frac{1}{poly}$, for many $\lambda)$

εnegligible:$\forall d : \lambda \geq \lambda_d : \varepsilon(\lambda) \geq \frac{1}{\lambda^d}$ $(\varepsilon \leq \frac{1}{poly}$, for large $\lambda)$

When we talk about probability of events, we talk about them as functions of a security parameter. These functions act on non-negative integers and output positive real values. So for the function to be non-negligible, it means that the function is bigger than some polynomial infinitely often. In other words for infinitely many values, the function is bigger than $\frac{1}{\text{some polynomial}}$.

If something is smaller than all polynomials then we say that it's negiligible. So what this says is that for any degree polynomial, d, there exists some lower bound λ_d such that for all λ bigger than that lambda (λ_d), the function ϵ is smaller than $\frac{1}{polynomial}$. So all this says is that the function is negiligible if it's less than all the polynomial fractions, $(\frac{1}{\lambda^d})$ for sufficiently large λ.

Examples:

$\varepsilon(\lambda) = \frac{1}{2^\lambda}$: negiligible, because for any constant, d, there is a sufficiently large lambda, such that $\frac{1}{2^\lambda} < \frac{1}{\lambda^d}$

$\varepsilon(\lambda) = \frac{1}{\lambda^{1000}}$: non-negiligible, because $\frac{1}{\lambda^{d=10000}} < \frac{1}{\lambda^{1000}}$

$\varepsilon(\lambda) \begin{cases} \frac{1}{2^\lambda} & \text{for odd } \lambda \\ \frac{1}{\lambda^{1000}} & \text{for even } \lambda \end{cases}$: non-negiligible, because if a function happens to be only polynomially small very often, that means that this event is already too large to be used in a real cryptosystem.

1.3 Attacks on stream ciphers and the one time pad

1.3.1 Attack 1: Two time pad is insecure.

If the key is used in more than one message, it's insecure.

$$c_1 \leftarrow m_1 \oplus PRG(k)$$

$$c_2 \leftarrow m_2 \oplus PRG(k)$$

Eavesdropper does:

$$c_1 \oplus c_2 \rightarrow m_1 \oplus m_2$$

Enough redundancy in English and ASCII encoding that

$$m_1 \oplus m_2 \rightarrow m_1, m_2$$

Real world examples:

- **Project Venona:** (1941-1946): Russians used OTP, the key was generated by a person throwing dice. This was labourious, so they used the same keys to encrypt more than one messages.
- **MS-PPTP:** (Windows NT): Client server communication protocol. The entire communication from client to server is considered a single string. Messages are cocatenated, then the stream is encrypted with a stream cipher using a key. The problem is the same thing is happening on the server side using the same OTP key. Need different keys for Client$\rightarrow$Server and Server$\rightarrow$Client, should have a pair of keys in the shared key.
- **802.11b WEP:** [7:20]
  Problem 1: In WEP, there's a client and an access point, both sharing a secret 104 big key, k, and when they want to transmit a message to one another, say client sends a plain text message, he first appends a CRC checksum and then the concatenation gets encrypted using a stream cipher, where they key is a concatenation of IV and a a long term key, k. IV is a 24 bit string, it's a counter that increments by 1 for every packet. This is for changing the key for every frame to enforce 1 key per message. They changed the key by prepending the IV and then IV is sent in the clear with the ciphertext. The problem is that IV repeats after $2^{24} \approx 16$M frames, meaning it has to cycle and we get a two time pad.
  Problem 2: On some 802.11cards IV resets to 0 after power cycle and so afterwards you'll be using a 0 concatenated key to encrypt the message.
  Problem 3: Avoid related keys. The keys aren't randomized, so the PRG key for frame has the same 104 bit suffix. Turns out that the PRG used in WEP, RC4, is not designed to be secure when you use so closely related keys. There's an attack covered by Fluhrer, Mantin and Shamir (2001), that shows that after about 10^6 frames, you can recover the key. Better attacks came out with about 40k frames are sufficient to get the key.

  Treat the frames as one long stream and then XOR using the PRG. With different key for every frame, like before, use the PRG again, take the long term key and feed it into PRG, then first segment could be used as key for frame 1, etc. Now keys have no relation.
- **Disk encryption**. A file on disk is broken into blocks, all blocks encrypted. Suppose a user modifies the file. After saving/reencrypting. An attacker looking at this, will see that just some small segment changed. Bad idea to use stream ciphers for disk encryption.

Summary:

Never use steram cipher key more than once.

Network traffic: negotiate new key for every session (e.g. TLS)

Disk encryption: typically do not use a stream cipher.

1.3.2 Attack 2: no itegrity (OTP is malleable)

All they do is try to provide confidentiality. It's easy to modify cipher text and have known effects on the text.

Suppose you have some message, m, that gets encrypted

$$m \xrightarrow{enc(\oplus k)} m \oplus k$$

An attacker can modify the cipher text

$$m \oplus p \xleftarrow{dec(\oplus k)} (m \oplus k) \oplus p$$

Modification to siphertext are undetected and have predictable impact on plaintext.

1.4 Real-world stream ciphers

1.4.1 RC4 (1987)

Takes a variable seed, used as a key for the stream cipher, which expands the key into a 2048 bits, which will be used as an internal state for the generator. Then it executes a loop, which every operation of outputs one byte of output.

Used in HTTPS and WEP

Weakesses:

1. Bias in initial output: $Pr[2^{nd}\, byte = 0] = \frac{2}{256}$, ignore 256 bytes of the output, all biased.
2. Prob. of (0, 0) is $\frac{1}{256^2} + \frac{1}{256^3}$
3. Related key attacks

1.4.2 CSS

Easy to implement in hardware, based on:

Linear feedback shift register (LFSR):

A register consisting of cells with one bit. There are taps into the cells, not all, these taps feed into XOR and at every clock cycle, the shift register shifts to the left, last bit falls off, first bit becomes the XOR of the previous bit.

seed = initial state of LFSR

Examples (all broken):

- DVD encryption (CSS): 2 LFSRs
- GSM encryption (A5/1, 2): 3 LFSRs
- Bluetooth (E0): 4 LFSRs

1.4.3 CSS Attack

CSS: seed = 5 bytes = 40 bits

40 bits because at the time DVD encryption was designed, US expo regulations only allowed for export of algorithms with up to 40 bit keys.

CSS uses 2 LFSRs: 17 bit and 25 bit.

They are seeded as follows

17 bit key start with 1 and concatenate with 2 first bytes of the key

25 bit key 1 || last 3 bytes of key

They generate 6 and 8 bit outputs, and go through a mod 256 adder + carry from previous block (?) and output one byte per round.

This is brakeable in $\approx 2^{17}$ time.

If the files are mpeg, if you know the prefix of the plaintext, say 20 bytes, then XOR the 2 things (?) together, you'll get the initial segment of the PRG. Then try all 2^{17} possible values of the first LFSR and run it for 20 bytes. We can take the output and subtract from the output of the LFSR and get the first 20 byte output of the second LFSR. Turns out that by looking at the LFSR sequence it's easy to tell if it came from the 25 bit LFSR, if it didn't, then the guess for the 17 bit LFSR was incorrect and move to the next guess till we hit the correct one. Then we can predict the remaining output of the CSS.

1.4.4 Modern Stream Ciphers: eStream (2008)

5 stream ciphers came out of that project.

$$\text{PRG:} \{0,\, 1\}^S \times R \to \{0,\, 1\}^n$$

R - nonce

$\{0,\, 1\}^s$- seed

$n \gg s$

Nonce: a non-repeating value for a given key. Unique value that never repeats as long as the key is the same.

$$E(k,\, m;\, r) = m \oplus PRG(k;\, r)$$

A property of the nonce is that the pair (k; r) is never used more than once. We can re-use the key, because (k, r) are unique.

1.4.5 Salsa 20 (software + hardware)

$$\{0,\, 1\}^{128\, or\, 256} \times \{0,\, 1\}^{64} \to \{0,\, 1\}^n \qquad \text{(max n=}2^{73} \text{ bits}$$

128 (or 256) bit seed and 64 bit nonce

$$\text{Salsa20}(k;\, r) := H(k;\, (r,\, 0))\, ||\, H(k,\, (r,\, 1))\, ||\, \ldots$$

Given the key and the nonce, generate a long pseudo-random sequence by using function H, which takes 3 inputs: the key, k, nonce, r, and the counter that increments from step to step.

How does the function H work?

We start by expanding the states into 64 bytes long (something), put constants τ_0 4 bytes, k 16 bytes, τ_1 4 bytes, r 8 byes, i (index) 8 bytes, τ_2 4 bytes, k 16 bytes, τ_3 4 bytes. If we sum all of them , we get 64 bytes. All this comes from spec.

Then we apply a 1:1 invertible function, designed to be fast on x86 (SSE2), mapping function, h, 10 times. This by itself isn't random due to h being invertible, so you XOR the inputs and the final output (of h) and then you get the 64 byte output.

1.5 PRG Security Definitions

Let

$$G : K \to \{0,\, 1\}^n \text{ be a PRG}$$

PRG with keyspace K that outputs an n bit string.

Goal: define what that means for the otuput of the generator to be indistinguishable from random.

$$[K \overset{R}{\leftarrow} K, \text{ output } G(K)]$$

is "indistinguishable" from a truly random string

$$r \overset{R}{\leftarrow} \{0,\, 1\}^n, \text{output } r$$

Uniform distribution can output any of $\{0,\, 1\}^n$ strings with equal probability but the generator can output a tiny subset. We're arguing that an adversary that can look at the output of a generator in the tiny set, can distinguish it from the output of the uniform distribution over the entire set.

1.5.1 Statistical Tests

Statistical test on $\{0, 1\}^n$:

an algorithm A s.t $A(x)$ outputs "0" or "1"

x is an n bit string

0 - not random

1 - random

Ex.:

1. #0(x) - number of 0s in string x

$A(x) = 1 \iff |\#0(x) - \#1(x)| \leq 10 * \sqrt{(n)}$

2. #00(x) - number of consecutive 0s in x

in a random string we'll expect to see that about 1/4th of the time

$A(x) = 1 \iff \#00(x) - \frac{n}{4}$

3. $A(x) = 1 \iff$ max-run-of-0(x) $\leq 10 \times \log_2(n)$

In the past, a fixed set of statistical tests for randomness would be run against an algo and if all output 1 then algo is random.

1.5.2 Advantage (of statistical tests)

Let $G : K \to \{0, 1\}^n$ be a PRG and A a statistical test on $\{0, 1\}^n$

Define:

An **advantage** of the statistical test A wrt. generator G is the difference between how likely is the statistical test to output 1 when we give it a pseudo-random output generated by the generator vs how likely is the statistical test to output 1 given a truly random string

$$ADV_{PRG}[A, G] := \left| \underset{k \overset{R}{\leftarrow} K}{Pr} [A(G(K)) = 1 - \underset{r \overset{R}{\leftarrow} \{0,1\}^n}{Pr} [A(r) = 1] \right| \in [0, 1]$$

if ADV close to 1 $\rightarrow$means that statistical test A behavied differently when we gave it pseudo-random inputs from when we gave it truly random input. A can distinguish G from rand.

if ADV close to 0 $\rightarrow$ test behaves same on both types of inputs, can't distinguish G from random.

A silly exmaple: $A(x) = 0 \Rightarrow ADV_{PRG}[A, G] = 0$, because if the test always outputs 0, then it'll never output 1 for any input.

Example:

Suppose $G : K \rightarrow \{0, 1\}^n$ satisfies $msb(G(k)) = 1$ for $\frac{2}{3}$ of keys in K (msb = most significant bit)

Define stat. test $A(x)$ as:

$$if[msb(x) = 1] \text{ output "1" else output "0"}$$

Then

$$\begin{aligned} Adv_{PRG}[A, G] &= |Pr[A(G(k)) = 1] - Pr[A(r) = 1]| \\ &= \left|\frac{2}{3} - \frac{1}{2}\right| = \frac{1}{6} \end{aligned}$$

Suppose we give the pseudo-random input. From definition of G, we know with probability $\frac{2}{3}$ the output will be 1. In a truly random input, the probability of msb being 1 is $\frac{1}{2}$. We can say that A breaks G with advantage $\frac{1}{6}$.

1.5.3 Secure PRGs: crypto definition

We say that $G : K \rightarrow \{0, 1\}^n$ is a **secure PRG** if;

$\forall$ "efficient" statistical tests A: $Adv_{PRG}[A, G]$ is negiligible

Are there provable secure PRGs? We don't know because then we'd be able to prove P$\neq$NP.

Easy fact: **a secure PRG is unpredictable**. Given a prefix of the output of the generator, it's impossible to predict the next bit of the output.

We show: PRG predictable $\Rightarrow$ PRG is insecure

Suppose A is an efficient algorithm s.t.

We feed it first i bits of the output of the generator G, we choose a random key from keyspace.

$$\underset{k \xleftarrow{R} K}{PR} [A(G(k)|_{1,\ldots,i}) = G(k)|_{i+1}] \quad = \quad \frac{1}{2} + \varepsilon$$

$$\text{for non-negiligible } \varepsilon \text{ (e.g. } \varepsilon = \frac{1}{1000})$$

Let's see that we can break the generator with this algorithm:

Define statistical test B as follows:

$$B(x) = \begin{cases} \text{if } A(x|_{1,\ldots,i}) = x_{i+1} & \text{output 1} \\ else & \text{output 0} \end{cases}$$

Suppose we give this test a truly random string r and ask what is the probability that this statistical test outputs 1.

$$r \xleftarrow{R} \{0,1\}^n \qquad Pr[B(r)=1] = \frac{1}{2}$$

For a random string the first $i+1$ bit is independent of the first i bits, so whatever A outputs is independent of what the $i+1$ bit of the string r is. So the probability that it's going to be equal to some random bit is $\frac{1}{2}$

Suppose now, that we give the test a pseudo-random string and ask how likely is it to output 1.

$$\begin{aligned} r \xleftarrow{R} K \qquad Pr[B(G(k)) &= 1] > \frac{1}{2} + \varepsilon \\ &\Rightarrow \quad Adv_{PRG}[B,G] > \varepsilon \end{aligned}$$

By definition of A, we know that if we give it the first i bits of the output of the generator, it'll predict the next bit with the probability $\frac{1}{2}+\varepsilon$. What this means the diff between the quantities for random and pseudo-random strings, it's $> \varepsilon$. That means if algorithm A is able to predict the next bit with advantage ϵ, algorithm B is able to distinguish the output of the generator with advantage ε. So if A is a good predictor, B is a good statistical test that breaks the generator. The contrapositive is that if G is a secure generator, then there aren't no good statistical tests and as a result no predictors and the generator unpredictable.

An unpredictable PRG is secure (Yao 1982)

Let $G : K \to \{0,1\}^n$ be PRG

Theorem: if $\forall_{i \in \{0,\ldots,n-1\}}$ PRG G is unpredictable at position i then G is a secure PRG.

If next-bit predictors cannot distinguish G from random then no statistical test can.

Ex:

Let $G : K \to \{0,1\}^n$ be a PRG such that from last $\frac{n}{2}$ bits of $G(k)$ it is easy to compute the first $\frac{n}{2}$ bits. G is predictable for some $i \in \{0,\ldots,n-1\}$.

If the last $\frac{n}{2}$ bits supply the first $\frac{n}{2}$ it means that G is not secure, and therefore predictible (by Yao).

More Generally

Let P_1 and P_2 be two distributions over $\{0,1\}^n$.

Def: We say that P_1 and P_2 are **computationally indistinguishable** (denoted $P_1 \underset{p}{\approx} P_2$) if $\forall$ "efficient" statistical tests A

$$\left| \underset{x \leftarrow P_1}{Pr}[A(x)=1] - \underset{x \leftarrow P_2}{Pr}[A(x)=1] \right| < negligible$$

Example: a PRG is secure if $\left\{ k \xleftarrow{R} K : G(k) \right\} \underset{p}{\approx} uniform(\{0,1\}^n)$

1.6 Stream ciphers are semantically secure

Theorem: given $G : K \to \{0,1\}^n$ is a secure PRG $\Rightarrow$ stream cipher E derived from G is semantically secure.

$\forall$ semantically secure adversary A, $\exists$ a PRG adversary B s.t.

$$Adv_{SS}[A,E] \leq 2 \times Adv_{PRG}[B,G]$$

Suppose you have a semantic security adversary A, what you'll then do is we'll build a PRG adversary B that satisfies the above inequality. If we know that if B is an efficient adversary, since G is a secure generator, we know that this advantage is negiligible. Because rhs is negiligible, the left side is too and so the adversary has a negiligible advantage in attacking the stream cipher.

Proof: [3:00]