Cryptography

Notes from Stanford's Coursera courses

May 21, 2017

Contents

Part I

Chapter 1

Overview and stream ciphers

1.1 Discrete probability

1.1.1 Probability distribution

Consider a finite set U. One example might be the set $U = \{0,1\}^n$, the set of all length 2 binary strings.

Definition 1.1.1 (Probability distribution)**.** *The probability distribution P over a finite set U is a function*

$$P\colon U \to [0,1]$$

such that

$$\sum_{x \in U} P(x) = 1$$

Two important example distributions are listed below:

- Uniform distribution:

$$\forall x \in U : P(x) = 1/|U|$$

- Point distribution at x_0:

$$P(x_0) = 1$$

$$\forall x \neq x_0 : P(x) = 0$$

A **distribution vector** contains the weights assigned to each element in the set U written as a vector. For example, given $U = \{0,1\}^3$, the distribution vector is

$$(P(000), P(001), \ldots, P(111))$$

and has length $2^3 = 8$.

1.1.2 Events

Definition 1.1.2 (Event)**.** *For a set* $A \subseteq U$: $\Pr[A] = \sum_{x \in A} P(x) \in [0,1]$. *Note that* $\Pr[U] = 1$. *This set* A *is called an event.*

For example, given $U = \{0,1\}^8$

$$A = \{\text{all } x \text{ in } U \text{ such that } \text{lsb}_2(x) = 11\} \subseteq U$$

The union bound

For events A_1 and A_2

$$\Pr[A_1 \cup A_2] \leq \Pr[A_1] + \Pr[A_2]$$

$$A_1 \cap A_2 = \emptyset \implies \Pr[A_1 \cup A_2] = \Pr[A_1] + \Pr[A_2]$$

As an example, consider

$$A_1 = \{\text{all } x \text{ in } \{0,1\}^n \text{ such that } \text{lsb}_2(x) = 11\}$$

$$A_2 = \{\text{all } x \text{ in } \{0,1\}^n \text{ such that } \text{msb}_2(x) = 11\}$$

The intersection of these events is non-zero, and we can say

$$\begin{aligned}\Pr[\text{lsb}_2(x) = 11 \text{ OR } \text{msb}_2(x) = 11] = \Pr[A_1 \cup A_2] &\leq \Pr[A_1] + \Pr[A_2] \\ &= \tfrac{1}{2}\end{aligned}$$

1.1.3 Random variables

Definition 1.1.3 (Random variable)**.** *A random variable* X *is a function* $X: U \to V$*, where* X *takes its values from the set* V*.*

Alternatively, a random variable X defines a set V and induces a distribution on V.

$$\Pr[X = v] \coloneqq \Pr[X^{-1}(v)]$$

The right hand side of the equation above describes the probability of the event that when we sample a random element in the universe, we fall into the pre-image of v under the function X.

For example, we can define the random variable X that represents the least significant bit of an n-bit set as follows:

$$X: \{0,1\}^n \to \{0,1\}; X(y) = \text{lsb}(y) \in \{0,1\}$$

For the uniform distribution on $\{0,1\}^n$

$$\Pr[X = 0] = \tfrac{1}{2}, \Pr[X = 1] = \tfrac{1}{2}$$

Here then, X defines a uniform distribution on the set $\{0,1\}$.

Definition 1.1.4 (Uniform random variable)**.** *Let* U *be a set, e.g.* $U = \{0,1\}^n$*. We write* $r \xleftarrow{R} U$ *to denote a uniform random variable over* U*.*

$$\forall a \in U: \Pr[r = a] = \tfrac{1}{|U|}$$

Formally, r is the identity function: $r(x) = x \forall x \in U$. For example, let r be a uniform random variable on $\{0,1\}^2$. Define X as

$$X = r_1 + r_2$$

We have defined the random variable X in terms of the uniform random variable r. X is now a function that can be expressed as follows:

$$X \colon \{0,1\}^2 \to [0,2]$$

where X also defines a distribution on $[0,2]$ such that $P(X = 0) = \frac{1}{4}$, $P(X = 1) = \frac{1}{2}$, and $P(X = 2) = \frac{1}{4}$.

Randomized vs deterministic algorithms

Deterministic algorithms always give the same output for a given input, which is not the case for randomized algorithms. Therefore, a randomized algorithm can be thought of having a random variable as an additional parameter.

For example, the deterministic case would be $y \leftarrow A(m)$, whereas the randomized case would be $y \leftarrow A(m; r)$ where r is some random variable.

This second expression also defines y as a random variable like so:

$$y \xleftarrow{R} A(m)$$

1.1.4 Independence

Events A, B are independent if

$$\Pr[A \text{ and } B] = \Pr[A] \cdot \Pr[B]$$

Random variables X, Y taking values in V are independent if

$$\forall a, b \in V \colon \Pr[X = a \text{ and } Y = b] = \Pr[X = a] \cdot \Pr[Y = b]$$

Theorem 1.1.1. *Y is a random variable over $\{0,1\}^n$, X is an independent uniform random variable on $\{0,1\}^n$. Then $Z \coloneqq Y \oplus X$ is a uniform variable on $\{0,1\}^n$.*

The following is a proof for the case $n = 1$.

Y	P
0	P_0
1	P_1

X	P
0	$1/2$
1	$1/2$

When considering the variables together, we can combine probabilities through multiplication since the variables are independent.

Y	P	P
0	0	$1/2 \cdot P_0$
0	1	$1/2 \cdot P_0$
1	0	$1/2 \cdot P_1$
1	1	$1/2 \cdot P_1$

$$\begin{aligned}\Pr[Z=0] &= \Pr[(X,Y)=(0,0) \text{ OR } (X,Y)=(1,1)] \\ &= \Pr[(X,Y)=(0,0)] + \Pr[(X,Y)=(1,1)] \\ &= \tfrac{P_0}{2} + \tfrac{P_1}{2} \\ &= \tfrac{1}{2}\end{aligned}$$

Since $\Pr[Z] = 1$, $\Pr[Z = 1] = \frac{1}{2}$ also. Therefore, Z is a uniform random variable.

Birthday paradox

Let $r_1, \ldots, r_n \in U$ be independent identically distributed random variables.

Theorem 1.1.2. *When* $n = 1.2 \cdot |U|^{1/2}$ *then* $\Pr[\exists i \neq j : r_i = r_j] \geq \frac{1}{2}$.

Consider $|U| = 365$, the number of possible birthdays a person might have. Assuming every birthday is equally likely, how many people would you need in a room for it to be likely that two share the same birthday? Using the theorem above, the answer is

$$n \simeq 1.2 \cdot \sqrt{365} \simeq 24$$

1.2 Stream ciphers 1: the one-time pad and stream ciphers

Definition 1.2.1 (Cipher)**.** *A cipher defined over* $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ *is a pair of "efficient" algorithms* (E, D) *where*

$$E : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$$

$$D : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$$

such that

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

$\mathcal{K}$ is the key space, $\mathcal{M}$ is the message space, and $\mathcal{C}$ is the ciphertext space. E is often randomized, whereas D is always deterministic.

1.2.1 One time pad

This is the first example of a "secure" cipher.

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

The key is a random bit string as long as the message.

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

This is a very fast algorithm, but it requires a key as long as the message itself. If you are capable of securely transmitting the key, then you may as well use that method to simply transmit the message. Although this limits its uses, the one time pad is secure.

Using information theory to show the one time pad is secret

When is a cipher perfectly secure? The basic idea is that the ciphertext should reveal no "information" about the message.

Definition 1.2.2 (Perfect secrecy)**.** *A cipher (E, D) over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ has perfect secrecy if*

$$\forall m_0, m_1 \in \mathcal{M} \text{ where } (\text{len}(m_0) = \text{len}(m_1)) \text{ and } \forall c \in \mathcal{C}$$

$$\Pr[E(k, m_0) = c] = \Pr[E(k, m_1) = c]$$

where k is uniform in $\mathcal{K}$ ($k \xleftarrow{R} \mathcal{K}$).

Given particular ciphertext no one can tell if the message is m_0, m_1, ... The most powerful adversary learns nothing about the message from the ciphertext. This means so-called ciphertext only attacks are impossible (though there are others).

Lemma 1.2.1. *The one time pad has perfect secrecy.*

The proof is fairly straightforward.

$$\forall m, c\colon \Pr[E(k, m) = c] = \frac{\#\text{ keys } k \in \mathcal{K} \text{ such that } E(k, m) = c}{|\mathcal{K}|}$$

Or in words the probability that the encryption of a particular message m with a key k will yield a particular ciphertext c is equal to the number of keys that map m to c divided by the total number of keys.

If this value is a constant for all m and c, then according to the previous definition the cipher E has perfect secrecy. Clearly, since $|\mathcal{K}|$ is a constant, if the numerator of the right hand side is a constant, then the cipher has perfect secrecy.

For the one time pad, there is exactly 1 key that maps a message to a particular ciphertext. This 1 is constant, so the one time pad does have perfect secrecy.

Theorem 1.2.1. *Perfect secrecy implies $|\mathcal{K}| \geq |\mathcal{M}|$.*

Hence the OTP is unfortunately the most convenient perfectly secret cipher.

1.2.2 Stream cipher

One idea of creating a more pragmatic cipher is by replacing the "random" key in the one time pad with a "pseudorandom" key.

A pseudorandom number generator (henceforth referred to as PRG) is a function

$$G\colon \{0,1\}^s \to \{0,1\}^n \text{ where } n \gg 5$$

The input set is the seed space, from which a key would be derived. The output must both look random and be deterministically computable.

The encryption and decryption algorithms would be as follows:

$$c = E(k, m) \coloneqq m \oplus G(k)$$
$$m = D(k, c) \coloneqq c \oplus G(k)$$

We know from Theorem 1.2.1 that this cannot have perfect secrecy, so we must use another definition of adequacy. This will be based on the qualities of the PRG.

Pseudorandom number predictability

The minimal property for security is that the PRG be unpredictable. Supposing a PRG was predictable, this might mean that given the first i bits of its sequence, one can compute the rest:

$$\exists i\colon G(k)|_{1,\ldots,i} \xrightarrow{\text{algorithm}} G(k)|_{i+1,\ldots,n}$$

This is unacceptable; some protocols have defined headers that would allow an attacker to deduce the entire message.

Definition 1.2.3 (Predictable PRG)**.** *$G\colon \mathcal{K} \to \{0,1\}^n$ is predictable if there exists an "efficient" algorithm $\mathcal{A}$ and there exists $1 \leq i \leq n-1$ such that*

$$\Pr[\mathcal{A}(G(k))|_{1,\ldots,i} = G(k)|_{i+1}] \geq \tfrac{1}{2} + \epsilon$$

for some "non-negligible" ϵ, where $k \xleftarrow{R} \mathcal{K}$.

So, if an algorithm can give a slight hint about the next bits in the PRG, then the PRG is predictable.

Definition 1.2.4 (Unpredictable PRG)**.**

$\forall i\colon$ no "efficient" algorithm can predict bit $(i+1)$ for "non-negligible" ϵ

Negligibility

In practice ϵ is a scalar and

- ϵ non-negligible: $\epsilon \geq 1/2^{30}$ (likely to happen over 1GB of data)
- ϵ negligible: $\epsilon \leq 1/2^{80}$ (won't happen over the lifetime of key)

In theory ϵ is a function $\epsilon\colon Z^{\geq 0} \to R^{\geq 0}$ and

- ϵ non-negligible: $\exists d\colon \epsilon(\lambda) \geq 1/\lambda^d$ ($\epsilon \geq 1/\text{polynomial}$ for many λ)
- ϵ negligible: $\forall d, \lambda \geq \lambda_d\colon \epsilon\lambda \leq 1/\lambda^d$ ($\epsilon \leq 1/\text{polynomial}$ for large λ)

Some examples are as follows:

$$\epsilon(\lambda) = \tfrac{1}{2^\lambda}\colon \text{ negligible}$$

$$\epsilon(\lambda) = \tfrac{1}{\lambda^{1000}}\colon \text{ non-negligible}$$

$$\epsilon(\lambda) = \begin{cases} 1/2^\lambda & \text{for odd } \lambda \\ 1/\lambda^{1000} & \text{for even } \lambda \end{cases} : \text{ non-negligible}$$

A bad PRG

This is a simple PRG with some good properties (the distribution of numbers is fairly even), but which is completely unsuitable for cryptography. It has parameters a, b, and a prime p.

$$
\begin{aligned}
r[0] \coloneqq\ & \text{seed} \\
& r[i] \leftarrow (a \cdot r[i-1] + b) \bmod p \\
& \qquad \text{output a few bits of } r[i] \\
& \qquad i\texttt{++}
\end{aligned}
$$

The `rand()` function in `glibc` was something similarly predictable.

1.3 Stream ciphers 2: attacks and common mistakes

1.3.1 Two-time pad attack

One major issue is that the two time pad is insecure. A one time pad (or stream cipher key) can never be used more than once.

$$
\begin{aligned}
c_1 &\leftarrow m_1 \oplus \mathrm{PRG}(k) \\
c_2 &\leftarrow m_2 \oplus \mathrm{PRG}(k)
\end{aligned}
$$

If an eavesdropper intercepts c_1 and c_2, they could then calculate:

$$c_1 \oplus c_2 = m_1 \oplus m_2$$

There is enough redundancy in English and ASCII encoding that m_1 and m_2 can then be recovered. This property has been exploited in the past.

MS-PPTP (Windows-NT)

In Microsoft's Point-to-Point Tunneling Protocol in Windows-NT, the setup was something like the following:

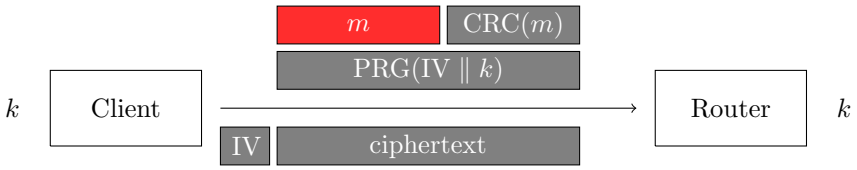
$[m_1 \parallel m_2 \parallel m_3 \parallel \ldots] \oplus \mathrm{PRG}(k)$ $\qquad$ $[s_1 \parallel s_2 \parallel s_3 \parallel \ldots] \oplus \mathrm{PRG}(k)$

The messages m_i were concatenated so that no two shared the same key, and the same was done for the responses s_i, but the same key was used for both messages *and* responses. Two separate keys should have been used, one for the client and one for the server (with both sides knowing both keys).

The arrangement as it was created a two-time pad, and so was insecure.

802.11b WEP

This wifi protocol is famously insecure and is now rarely used. It works according to the diagram below:



The diagram shows the client sending a packet m to the router with a checksum concatenated with it. The data is encrypted with a stream cipher, where the stream cipher key is the concatenated of a long-term key k and a 24-bit string IV. This string is sent in plain text so the router knows what it is.

Each time a packet is sent, IV is incremented, but since there are only 2^{24} values of IV, it is repeated after around 16M frames (not that many in a busy network). This leads to the same IV being used to encrypt two different messages which, along with the non-changing nature of the long term key k, leads to a two-time pad.

Worse, IV resets to 0 after a power cycle on many 802.11 cards, exacerbating the problem.

An even more major issue with WEP is that the IV string was not randomized, and changed only slightly in a predictable way with each new packet. Since all the resultant keys were so close to each other, and because the PRG used in WEP (Rc4) was not designed to compensate for this, deductions could be made about the long-term key k.

An attack discovered in 2001 by Fluhrer, Mantin, and Shamir showed that it was possible after only around 10^6 frames to recover the secret key k. Since then more attacks have been discovered that show that the related nature of the generated keys was so disastrous that after only around $40,000$ frames the master key can be recovered.

How WEP could have been designed better One possibility would have been to treat the separate frames as one long stream and xor'd them using one long pad generated by the PRG. If having a different key for every frame was important however, then instead of slightly modifying the key k each time, k could have been used as a seed to another PRG. The output of this would then be the actual keys used.

Disk encryption

A final example of the dangers of the two-time pad would be using stream ciphers in disk encryption.

Imagine a file that has been encrypted and then saved on disk. If an attacker acquired a snapshot of the disk, and the file was then modified slightly, only some of the encrypted file would change. The attacker would then be able to see where this change was and use the fact that both files were encrypted with the same key to deduce the changes made.

Ideally, if a file is changed, the entire contents of the encrypted file should change. This is not the case with a stream cipher (or any one-time pad). At the very least any change to a file should completely change the contents of the encrypted block of data where the change took place. With the stream cipher an attacker can tell not only which block was changed (files are stored in blocks on disk), but also where within the block.

1.3.2 Another attack: no integrity

Not only can stream ciphers be vulnerable to two-time pad attacks, but they also offer no integrity. This means that one can modify ciphertext and have known effects on the plain text. This property is called **malleability**.

Consider a standard one time pad encryption:

$$c = m \oplus k$$

A perturbation p can be added to the resultant ciphertext that will, when the message is decrypted, produce a known effect on the plaintext.

$$c \oplus p = (m \oplus k) \oplus p$$

Decrypting this modified ciphertext produces:

$$((m \oplus k) \oplus p) \oplus k = m \oplus p$$

Example Let's assume Bob sends a message to Alice. He encrypts it using a stream cipher, however Eve intercepts the message and wishes to make it appear to have been sent by her. Eve may know that the message contains the text `From: Bob`.

If Eve knows the location of this text, she can modify the ciphertext to make it so that the decrypted message appears to be from Eve.

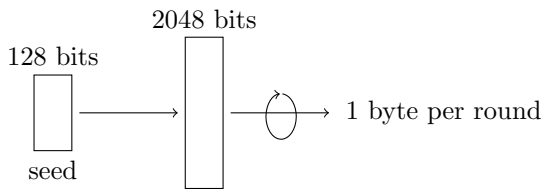
Consider that `Bob` in ASCII is `42 6F 62` and `Eve` is `45 76 65`. Eve can simply xor the ciphertext with `Bob` $\oplus$ `Eve` = `07 19 07` so that Alice decrypts a message that reads `From: Eve`.

Having the ability to predictably impact the ciphertext (malleability) can be a serious issue. The one time pad by itself has no integrity.

1.4 Stream ciphers 3: real world examples

1.4.1 RC4

RC4 was designed in 1987, and is not recommended for use any more. It takes a variable-sized seed (e.g. 128 bits) and expands to a 2048 bit internal state. After this, a simple loop function is executed that provides one byte of output at a time. This loop can be run indefinitely to keep producing output.



RC4 was used in WEP (incorrectly) and is still used in some HTTPS implementations.

Weaknesses

There are three broad weaknesses in RC4:

- Bias in initial output: $\Pr[2^{\text{nd}} \text{ byte} = 0] = \frac{2}{256}$ (should be $\frac{1}{256}$)
  - Output should only be used after first 256 bytes have passed
- Probability of $(0, 0)$ in a long output is $\frac{1}{256^2} + \frac{1}{256^3}$
  - This only starts after several GB of output
- Related key attacks (using keys closely related to one another makes it possible to recover the key)

1.4.2 CSS and LFSRs

The Content Scrambling System used to encrypt DVDs is now badly broken. It uses a stream cipher based on a Linear Feedback Shift Register (LFSR), which is easy to implement in hardware.

The register has cells where each cell contains one bit. Certain cells have so-called 'taps' which take their values and feed into an xor. Every clock cycle the shift register shifts to the left, the last bit falls off, and the first bit becomes the xor of the sampled bits.

The seed for the LFSR is simply its initial state. The system is shown in Figure 1.1

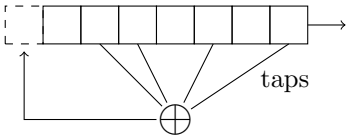


Figure 1.1: working of an LFSR

DVD encryption (CSS) uses 2 LFSRs, GSM encryption (A5/1,2) uses 3 LFSRs, and Bluetooth (E0) uses 4LFSRs. These happen to all be quite badly broken, but since they are implemented in hardware they are hard to change now.

CSS

CSS was limited to having a 5 byte (40 bit) seed by the US limits on crypto exports. It uses 2 LFSRs. The first is a 17-bit LFSR (the register contains 17 bits), and the second is a 25-bit LFSR.

The way these are seeded is as follows:

- The first 17-bit LFSR is seeded by a 1 concatenated with the first 2 bytes of the key

- The second 25-bit LFSR is seeded by a 1 concatenated with the last 3 bytes of the key

The stream cipher pad is then produced from cycling each LFSR 8 times, then taking these two 8-bit numbers and adding them modulo 256 (there is also one additional bit added here, the 'carry' from the previous block, but this isn't hugely relevant).

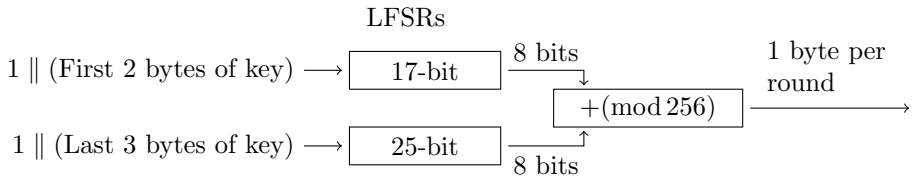


Figure 1.2: the PRG from CSS

The output is then xor'd with the relevant byte of the film being encrypted.

Breaking CSS

CSS may be fast and easy to implement, but it is also easy to break in time $\sim 2^{17}$. Because CSS is used to encrypt MPEG files, the first, say, 20 bytes of the file is known.

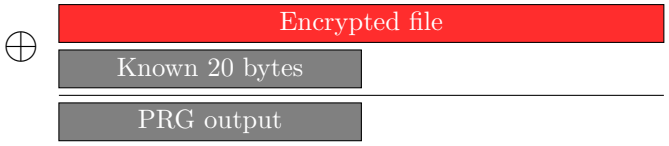


Figure 1.3: breaking CSS

By taking the xor of the first 20 bytes of an MPEG file with the first 20 bytes of the encrypted DVD, the first 20 bytes of the output of the PRG can be found (as shown in Figure 1.3).

Given this, all 2^{17} values of the first LFSR are used to generate 20 bytes of output. The PRG output gained in the previous step is then subtracted from each generated 20 bytes of output in turn. One of these guesses would then correspond to the first 20 bytes of output from the second LFSR (the guess that happened to be the correct key for the first LFSR). This can be seen by observing the way the LFSR outputs are combined to form the PRG (Figure 1.2).

As it happens, it is very easy to tell whether a particular 20 byte sequence came from the 25-bit LFSR or not. So when the correct key for the first 17-bit LFSR is found, it is easy to know.

This means that the problem is reduced to simply trying 2^{17} values for first 2 bytes of the key, after which both initial LFSR states can be recovered. Given this, the rest of the film can be decrypted.

1.4.3 Better stream ciphers: eStream

The eStream project (concluded in 2008) qualified 5 different stream ciphers. These ciphers take two inputs, a seed and a 'nonce'. The nonce is a non-repeating value for a given key.

$$\text{PRG}\colon \{0,1\}^s \times R \to \{0,1\}^n \text{ where } n \gg s$$

R here is the nonce. The encryption algorithm E is then:

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

The pair (k, r) is never used more than once. You can then reuse the key, because the nonce r makes the pair unique.

Salsa20

Salsa20 is one eStream cipher designed to be easy to implement in hardware *and* fast to implement in software on an x86 chip (due to its taking advantage of SSE2 instructions).

$$\text{Salsa20}\colon \{0,1\}^{128 \text{ or } 256} \times \{0,1\}^{64} \to \{0,1\}^n \text{ (max } n = 2^{73} \text{ bits)}$$

The actual algorithm works as follows:

$$\text{Salsa20}(k; r) \coloneqq H(k, (r, 0)) \parallel H(k, (r, 1)) \parallel \ldots$$

where H is a function that is defined to be 10 iterations of another function h on a 64 byte input as shown in Figure 1.4. h is a one-to-one invertible function, and the τ_i's shown are fixed 4 byte constants.

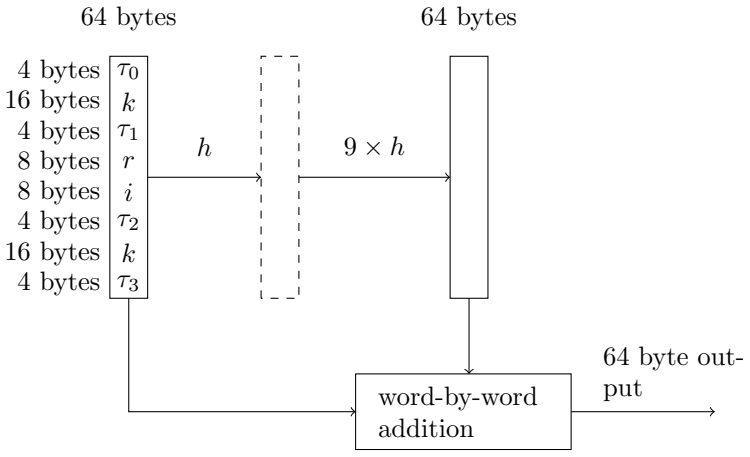


Figure 1.4: diagram of the function H

Without the word-by-word (that is, 4 bytes at a time) addition at the end, it would be easy to invert the output and go back the original input (since h is invertible). As it is, there are no significant attacks on this algorithm yet.

Comparing stream cipher generators

Comparing PRGs, it can be seen that the modern eStream versions are not only more secure, but also faster than RC4:

PRG	Speed (MB/sec)
RC4	126
Salsa20/12	643
Sosemanuk	727

The last two ciphers here are from the eStream project, and are more than fast enough to, say, decrypt a film.

1.5 Stream ciphers 4: what is a secure cipher?

Consider a PRG with key space $\mathcal{K}$ that outputs n-bit strings.

$$G\colon \mathcal{K} \to \{0,1\}^n$$

What does it mean for the output of the generator to be indistinguishable from random? Or in mathematical notation, what does it mean for

$$[k \xleftarrow{R} \mathcal{K},\ \text{output } G(k)]$$

to be indistinguishable from

$$[r \xleftarrow{R} \{0,1\}^n,\ \text{output } r]$$

It is perhaps surprising that any PRG can seem indistinguishable, considering the key space is considerably smaller than the space of n-bit strings that true randomness can draw from.

1.5.1 Statistical tests

Definition 1.5.1 (Statistical test)**.** *An algorithm A such that $A(x)$ outputs* 0 *or* 1 *based on whether the input x is not random or random respectively.*

Examples:

- $A(x) = 1$ iff $|\#0(x) - \#1(x)| \leq 10 \cdot \sqrt{n}$
- $A(x) = 1$ iff $\left|\#00(x) - \frac{n}{4}\right| \leq 10 \cdot \sqrt{n}$
- $A(x) = 1$ iff max-run-of-0$(x) \leq 10 \cdot \log_2(n)$
- Many others...

Each of these will output 1 with a high probability when encountering random output.

Advantage

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

Definition 1.5.2 (Advantage)**.** *The advantage of a PRG is defined according to*

$$[A,G] \coloneqq \left| \Pr_{k \xleftarrow{R} \mathcal{K}}[A(G(k)) = 1] - \Pr_{r \xleftarrow{R} [0,1]^n}[A(r) = 1] \right| \in [0,1]$$

If the advantage is close to 1 then A can distinguish output of G from random. If the advantage is close to 0 then A performs similarly on pseudorandom output to random output, and so it cannot distinguish the two.

Small example As a simple example, the following test A can never distinguish between anything:

$$A(x) = 0$$

$$\mathrm{Adv}_{\mathrm{PRG}}[A,G] = 0$$

More interesting example Suppose $G\colon \mathcal{K} \to \{0,1\}^n$ satisfies $\mathrm{msb}(G(k)) = 1$ for $2/3$ of keys in $\mathcal{K}$.

Define a statistical test $A(x)$ as:

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

Then:

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

Since this advantage is non-negligible the statistical test A *can* distinguish between G and true randomness.

1.5.2 Secure PRG

Definition 1.5.3 (Secure PRG)**.** *$G\colon \mathcal{K} \to \{0,1\}^n$ is secure if*

$$\forall \text{ ``efficient'' statistical tests } A\colon \mathrm{Adv}_{PRG}[A,G] \text{ is ``negligible''}$$

The requirement for "efficiency" here is necessary. If this were removed, the definition would be unsatisfiable. The question then is whether there are any provably secure PRGs. In fact, it is unknown whether this is the case.

It turns out that if $\mathbf{P} = \mathbf{NP}$ there are *no* secure PRGs. Hence finding one would prove $\mathbf{P} \neq \mathbf{NP}$.

Lemma 1.5.1. *A secure PRG is unpredictable.*

To show this, we show that a predictable PRG implies that the PRG is insecure.

Suppose A is an "efficient" algorithm such that

$$\begin{aligned} A_{k \xleftarrow{R} \mathcal{K}}(G(k)|_{1,\ldots,i}) &= G(k)|_{i+1} \\ &= \tfrac{1}{2} + \epsilon \end{aligned}$$

For non-negligible ϵ, this mean that G is predictable. We can then define a statistical test B such that

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

- For $x = r \xleftarrow{R} \{0,1\}^n$
  - $\Pr[B(r) = 1] = \frac{1}{2}$
- For $x = k \xleftarrow{R} \mathcal{K}$
  - $\Pr[B(G(k)) = 1] > \frac{1}{2} + \epsilon$

Hence the advantage of B over G is greater than ϵ. This implies that G is insecure. So if A is a good predictor, B is a good statistical test that breaks G.

This proves Lemma 1.5.1, that a secure PRG is unpredictable.

Theorem 1.5.1 (Yao 1982)**.** *An unpredictable PRG is secure. Let* $G\colon \mathcal{K} \to \{0,1\}^n$ *be a PRG*

If $\forall i \in \{0,1,\ldots,n-1\}$ *PRG* G *is unpredictable at pos.* i *then* G *is secure*

Theorem 1.5.1 is the converse of Lemma 1.5.1. We won't prove this here, but the implication is that if next-bit predictions cannot distinguish G from random then no statistical test can!

Example As an example, let $G\colon \mathcal{K} \to \{0,1\}^n$ be a PRG such that from the last $\frac{n}{2}$ bits of $G(k)$ it is easy to compute the first $\frac{n}{2}$ bits. Is G predictable for some $i \in \{0,1,\ldots,n-1\}$?

The answer is YES, which implies by Yao's theorem that G is not secure.

Computational indistinguishability

Let P_1 and P_2 be two distributions over $\{0,1\}^n$. Define P_1 and P_2 to be computationally indistinguishable (denoted $P_1 \approx_P P_2$) if $\forall$ "efficient" statistical tests A,

$$|\Pr_{x \leftarrow P_1}[A(x) = 1] - \Pr_{x \leftarrow P_2}[A(x) = 1]| < \epsilon$$

So a PRG is secure if

$$\{k \xleftarrow{R} \mathcal{K}\colon G(k)\} \approx_P \text{uniform}(\{0,1\}^n)$$

1.5.3 Secure ciphers and semantic security

Given the ability of an attacker to obtain one ciphertext (for now). What are some possible security requirements?

- Attacker cannot recover secret key
  - Bad - $E(k,m) = m$ would be considered secure under this definition
- Attacker cannot recover all of the plaintext
  - Bad - $E(k, m_0 \parallel m_1) = m_0 \parallel E(k, m_1)$ would be considered secure under this definition

Recall Shannon's idea: the ciphertext should reveal no "information" about the plaintext. Remember Shannon's perfect secrecy; let (E, D) be a cipher over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, then (E, D) has perfect secrecy if $\forall m_0, m_1 \in \mathcal{M}(|m_0| = |m_1|)$

$$\{E(k, m_0)\} = \{E(k, m_1)\} \text{ where } k \xleftarrow{R} \mathcal{K}$$

where the curly braces denote the distribution of ciphertexts.

If the adversary observes any ciphertext, there is no way to know whether it came from m_1 or m_2 for all m_i of the same length. This definition is too strong since it requires $|\mathcal{K}| \geq |\mathcal{M}|$. Instead we can say that (E, D) has secrecy if $\forall m_0, m_1 \in \mathcal{M}(|m_0| = |m_1|)$

$$\{E(k, m_0)\} \approx_P \{E(k, m_1)\} \text{ where } k \xleftarrow{R} \mathcal{K}$$

In other words, we only require that the two distributions are computationally indistinguishable, or that an efficient attacker cannot distinguish between them.

It turns out that this definition is still a little two strong, so one more constraint must be added. Rather than requiring this $\forall m_0, m_1 \in \mathcal{M}$, we can relax it to only those $m_0, m_1 \in \mathcal{M}$ that the adversary can exhibit explicitly.

Semantic security for a one-time key

For $b = 0, 1$ define experiments EXP(0) and EXP(1) as:

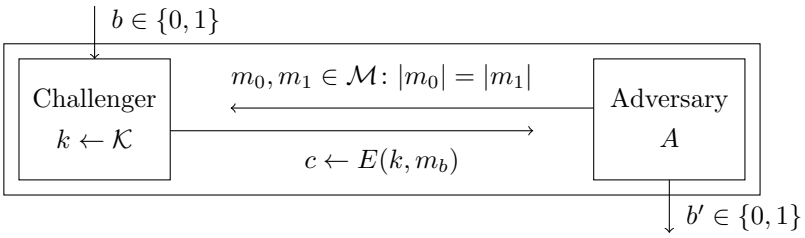


Figure 1.5: diagram of experiment to prove semantic security

The experiments consist of a challenger and an adversary. The challenger picks a random key k, and the adversary will give out two messages of equal length m_0 and m_1. The challenger will then output the encryption of m_b depending on the input to the experiment, and the adversary will attempt to guess the value of b and output this guess b'.

We define the event W_b to be the event that in experiment b, the adversary output 1:

$$\text{For } b = 0, 1 \colon W_b \coloneqq [\text{event that } \text{EXP}(b) = 1]$$

Definition 1.5.4 (Semantic security advantage)**.** *The semantic security advantage of the adversary A against the encryption scheme E is*

$$\text{Adv}_{SS}[A, E] \coloneqq |\Pr[W_0] - \Pr[W_1]| \in [0, 1]$$

In other words, the question is whether the adversary behaves differently when given the encryption of m_0 to when given the encryption of m_1. In experiment 0 the adversary is given the encryption of m_0 and in experiment 1 they were given the encryption of m_1. We are simply interested in whether the adversary output 1 or not. If in both experiments the adversary output 1 with the same probability, then the adversary could not distinguish the two experiments (and so the semantic security advantage would be 0).

Definition 1.5.5 (Semantic security)**.** $\mathbb{E}$ *is semantically secure if for all "efficient" adversaries* A

$$\mathrm{Adv}_{SS}[A, \mathbb{E}] \text{ is "negligible"}$$

which implies that for all explicit $m_0, m_1 \in \mathcal{M}$*:*

$$\{E(k, m_0)\} \approx_P \{E(k, m_1)\}$$

Example

Suppose an "efficient" adversary A can always deduce the LSB of the plaintext from a ciphertext. We can show using Definition 1.5.5 that $\mathbb{E} = (E, D)$ is not semantically secure. This will then imply that a semantically secure system would disallow attacks of this type.

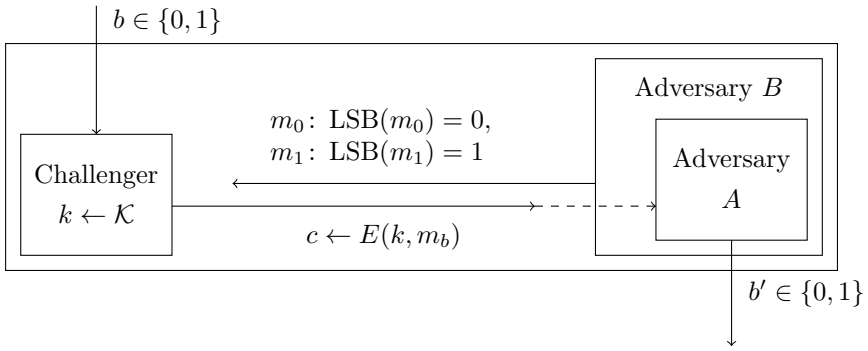


Figure 1.6: an example of an adversary with a large semantic security advantage

If adversary A simply outputs the least significant bit of the message m_b, then that will mean that the output b' will always equal b (since LSB$(m_b) = b$). This means that the advantage

$$\begin{aligned} \mathrm{Adv}_{SS}[B, \mathbb{E}] &= |\Pr[\mathrm{EXP}(0) = 1] - \Pr[\mathrm{EXP}(1) = 1]| \\ &= |0 - 1| \\ &= 1 \end{aligned}$$

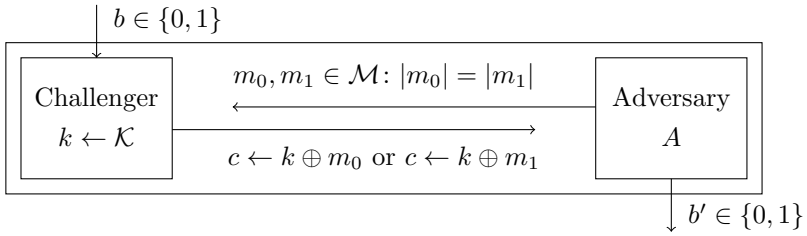
which means that the adversary completely broke the semantic security of the cipher. This means that $\mathbb{E}$ is *not* semantically secure.

Clearly this would also hold if any one bit of the message was deductible, or indeed any 'bit' of information about the message. Hence if a cipher is semantically secure, then no 'bit' of information is revealed to an adversary.

This is really just perfect security applied to efficient, rather than all, adversaries.

Showing the one-time pad is semantically secure

Clearly the OTP is semantically secure, since we already know it to be perfectly secure. But to show this we could consider an experiment as follows:



Since $k \oplus m_0$ is distributed *exactly* the same as $k \oplus m_1$ we can say

$$\begin{aligned} \forall A\colon \mathrm{Adv}_S\, S[A, \mathbb{E}] &= |\Pr[A(k \oplus m_0) = 1] - \Pr[A(k \oplus m_1) = 1]| \\ &= 0 \end{aligned}$$

and hence the OTP is semantically secure. In fact, it is secure for all adversaries, not just efficient ones.

1.5.4 Stream ciphers are semantically secure

Now we can show that a stream cipher with a secure PRG is semantically secure.

Theorem 1.5.2.

$$G\colon \mathcal{K} \to \{0,1\}^n \textit{ is a secure PRG}$$

implies that a stream cipher $\mathbb{E}$ *derive from* G *is semantically secure.*

There is no hope of proving this theorem for perfect secrecy, since we know a stream cipher cannot be perfectly secure (since it has keys shorter than the message).

To prove Theorem 1.5.2 we will show the following lemma:

Lemma 1.5.2. $\forall$ *semantic security adversaries* A, $\exists$ *a PRG adversary* B *such that*

$$\mathrm{Adv}_{SS}[A, \mathbb{E}] \leq 2 \cdot \mathrm{Adv}_{PRG}[B, G]$$

If this holds it would imply that the left hand side is negligible, since B is an efficient adversary and G is a secure generator (meaning the right hand side is negligible).

So given any adversary A, we will build an adversary B which we know has negligible advantage over G, and show that the advantage of A over $\mathbb{E}$ is also negligible.

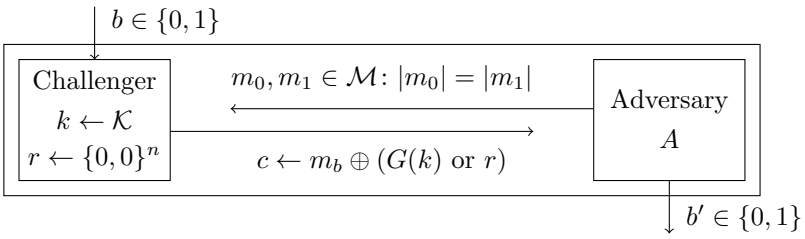


Figure 1.7: experiment on a stream cipher

Figure 1.7 shows two different situations; the first where the ciphertext is encrypted using the secure generator G:

$$\text{For } b = 0, 1 \colon W_b \coloneqq [\text{event that } b' = 1]$$

and the second using a truly random pad r:

$$\text{For } b = 0, 1 \colon R_b \coloneqq [\text{event that } b' = 1]$$

Remember that

$$\text{Adv}_{\text{SS}}[A, \mathbb{E}] = |\Pr[W_0] - \Pr[W_1]|$$

Proof of Lemma 1.5.2 Let A be a semantic security adversary. When we switch from the generator G to the one-time pad, the adversary can't tell, because G is computationally indistinguishable from randomness. This already informally proves the semantic security of $\mathbb{E}$, but we can be more formal.

- Claim 1: $|\Pr[R_0] - \Pr[R_1]| = \text{Adv}_{SS}(A, \text{OTP}) = 0$
  - This is obvious, since we already know the OTP has perfect secrecy
  - It also means that $\Pr[R_1] = \Pr[R_2]$
- Claim 2: $\exists B \colon |\Pr[W_b] - \Pr[R_b]| = \text{Adv}_{\text{PRG}}(B, G)$ for $b = 0, 1$
  - The implication of this being true is shown in Figure 1.8: Lemma 1.5.2 holds and so proves Theorem 1.5.2

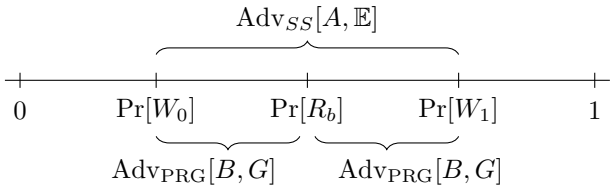


Figure 1.8: relation between PRG advantage and SS advantage

If the second claim holds, Figure 1.8 shows that the distance between $\Pr[W_0]$ and $\Pr[W_1]$ is at most twice the advantage of B against G.

Proof of claim 2 We can construct an appropriate statistical test B by considering one that uses adversary A inside of it as shown in Figure 1.9.

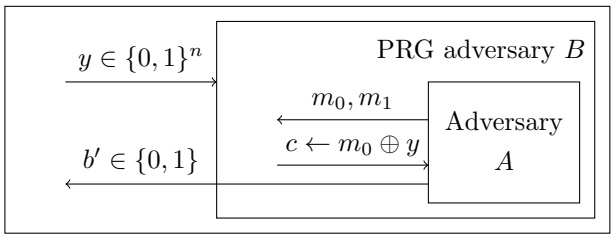


Figure 1.9: another algorithm

We can now look at the advantage of B over G as follows:

$$\begin{aligned} \mathrm{Adv}_{\mathrm{PRG}}[B,G] &= \left|\Pr_{r\leftarrow\{0,1\}^n}[B(r)=1] - \Pr_{k\leftarrow\mathcal{K}}[B(G(k))=1]\right| \\ &= |\Pr[R_0] - \Pr[W_0]| \end{aligned}$$

And so we have, given an adversary A, constructed a test B such that claim 2 holds. This proves Lemma 1.5.2, which means that the advantage of A over $\mathbb{E}$ for any A is negligible, proving Theorem 1.5.2.

Chapter 2

Block ciphers

Chapter 3

Message integrity

Chapter 4

Authenticated encryption

Chapter 5

Basic key exchange

Chapter 6

Public-key encryption

Part II