Cryptography

A.Y. 2024/2025

Contents

1. Introduction

In a specific scenario (referred to as symmetrical one) in which two people want to communicate secretly, they share both an algorithm and a secret key (the latter is not shared with anyone else). These protocols are specifically designed to ensure confidentiality during data exchange between two parties.

The encryption scheme consists of three spaces $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ and a triple of algorithms:

- **Generation**: while not a true mathematical function, it can be represented as $\mathsf{Gen} : 1 \mapsto \mathcal{K}$, where it generates a key. It's often referred to as random-function of random-algorithm.
- **Encryption**: it's a deterministic algorithm $\mathsf{Enc} : \mathcal{M} \times \mathcal{K} \mapsto \mathcal{C}$.
- **Decryption**: similarly, it's represented as $\mathsf{Dec} : \mathcal{C} \times \mathcal{K} \mapsto \mathcal{M}$.

The scheme is correct if $\mathsf{Dec}(\mathsf{Enc}(x, k), k) = x$ but, correctness does not assure security (for instance, the identity function is correct but not secure).

Kerckoff's principle

The only secret part of the communication is the key. Both participants in the communication are fully aware of the entire schema.

1.1. Possible attacks

- Ciphertext-Only

The attacker knows a certain number of ciphertexts $c_1, ..., c_n$ so it stays in a passive way and interferes with the communication reading the sent ciphertext.

- Known-Plaintext

Another passive attack where the attacker knows a certain number of pairs $(m_1, c_1), ..., (m_n, c_n)$. So, the attacker can understand the original m_i by the intercepted c_i.

- Chosen-Plaintext

In this case the attacker plays an active role computing $\mathsf{Enc}_k(m) = \mathsf{Enc}(m, k)$ for a message of their choice. Those who know k, know everything.

- Chosen-Ciphertext

The attacker has access to an oracle for the decryption $\mathsf{Dec}_k(\cdot)$ without having access to the key k.

1.2. Some historical ciphers

- Caesar

It is a good example of what we do not want to do. This is one of the best ones and more famous as example of symmetrical cipher. We just need a message $\mathcal{M}$ and a single key in a set $\mathcal{K}$, e.g. $\mathcal{K} = \{4\}$.

Since the Enc algorithm takes the key and shifts the message of k characters, the only thing an attacker might do is, given a ciphertext, shifting the message of $-k$ characters (i.e. backwards).

The alphabet used here is composed by the letters of the english alphabet.

- Shift

It's a generalisation of the cipher above: the alphabet can be made by different characters. The number of positions you shift is part of the key.

For instance, given a message `S | I | O | R | D` the ciphertext will be `S+n | I+n | O+n | R+n | D+n` of course the shifter goes back for characters $>$ |alphabet|. The $\mathcal{K}$ becomes $\{1, ..., |\Sigma| - 1\}$.

An attacker can try to decrypt a ciphertext with all the possible keys with at most $|\Sigma| - 1$ attempts.

For different data types (such as images) is not useful.

- Mono-alphabetic substitution

Another generalisation but for both ciphers above.

$\mathcal{K}$ becomes $\{\sigma \mid \sigma : \Sigma \mapsto \Sigma \text{ is a permutation}\}$.

So, there is a function which maps the character to another. Indeed, the previous message example of `S | I | O | R | D` has a ciphertext made by $\sigma(S)$ | $\sigma(I)$ | $\sigma(O)$ | $\sigma(R)$ | $\sigma(D)$.

You need to use an injective function.

A brute-force attack is impossible here, because $|\mathcal{K}|$ is $|\Sigma|!$.

- Vigenère

We do not apply the same transformation for each character. Each char uses different keys.

The previous example `S | I | O | R | D` has a ciphertext made by $S + B_1$ | $I + B_2$ | $O + B_3$ | $R + B_4$ | $D + B_5$.

- Take-home messages

It's considered not secure but one of the most famous in history. We have a large enough key to prevent brute-force attack (this is necessary, not sufficient). But, the length of a key does not assure a prevention at all.

1.3. Statistical attacks

Ciphertexts shown until now are not secure. The idea of this attack is to analyse the frequencies of the symbols in the ciphertext comparing the numbers with the numbers you know about the message. This makes sense if you're considering a natural language message where you know about the frequencies of a character and words in that natural language: knowing these frequencies you can easy understand the mapping of a letter. (eg: you know that `the` is the most frequent word in an english sentence).

In other words, given the frequencies $q_1, ..., q_{|\Sigma|}$ for each symbol in the ciphertext c you can build tables through the probabilities p_i where i-th is the sentence of that language. As $|c|$ increases, the probability of success converges to 1.

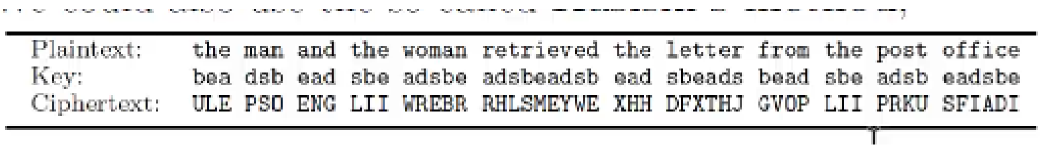
This statistical attack for Shift cipher can be computed as

$K = \sum_{i=1}^{|\Sigma|} p_i^2$

$I_j = \sum_{i=1}^{|\Sigma|} (p_i \cdot q_{i+j})$

If we encrypt data for images does not mean it'll also be valid for a text.

There is another kind of the same attack but for the Vigenère cipher called **Kasiski's method**. The observation here is to understand the period of the key (or better, the number of the characters of the key). Of course, you start knowing that a natural language has a fixed structure on words for bigrams and trigrams. A word in the plaintext could be encrypted with the same key in another point on the plaintext.



Plaintext:	the man and the woman retrieved the letter from the post office
Key:	bea dsb ead sbe adsbe adsbeadsb ead sbeads bead sbe adsb eadsbe
Ciphertext:	ULE PSO ENG LII WREBR RHLSMEYWE XHH DFXTHJ GVOP LII PRKU SFIADI

Figure 1: Vigenère statical attack

In Figure 1 you can see that `THE` is encrypted twice to `LLI` using the same key `SBE`. If you know this you can find out the length of the key and then (ya, working a lot more ofc) whole the key. The same trigrams are encrypted the same.

This kind of attack is computational simple because you just count.

There's also a method based on the index of coincidence to find out the length of the key. For increasing values of τ natural language we tabulate the characters of the ciphertext in position $1, 1 + \tau, 1 + 2\tau, 1 + 3\tau, \ldots$ obtaining the frequencies q_i^τ.

$K = \sum_{i=1}^{|\Sigma|} p_i^2$

$S_\tau = \sum_{i=1}^{|\Sigma|} (q_i^\tau)^2$

1.4. The 3 principles of modern cryptography

1. Use of rigorous and precise definitions of security of primitives and protocols

We need formal mathematical definitions.

2. Accuracy in specifying the underlying assumptions

We can't prove without assumptions.

3. Proof of security written in the language of math

We use the math (such as combinatory, statistics, …) to prove the security of an algorithm. This secures us to prove any possible to do something against the algorithm (thank you math `<3`) and not only once for a given example.

We need to formulate exact definitions during the design (you need to know what the goal is), the use (you should already know the limitations of an encryption scheme), the study and the compare (we can use an encryption scheme depending on its security issues comparing to another one).

We can't use inaccurate assumptions, even if they're not proved. Example of assumptions used by everyone are NP problems: no one really knows how to say if a problem is NP or not, but it's ok.

2. Perfectly-secret encryption

In a context of perfect-secrecy the Enc algorithm is possibly probabilist but Dec is deterministic for sure. We want correctness, and this is much more important than everything else.

The probabilistic process is a base for the random variables. We need 3 random variables:

- K that corresponds to the used key depending on Gen algorithm;
- M that corresponds to the message produced by the sender;
- C that corresponds to the ciphertext, dependant on K, M and Enc.

Then, we can calculate $\Pr(\boldsymbol{K} = k)$ with $k \in \mathcal{K}$ is a key but we also can calculate $\Pr(\boldsymbol{K} = k \mid \boldsymbol{M} = m)$ with $m \in \mathcal{M}$. The latter is a condition: you can evaluate the key if you know a message.

Key and message are always independent: two separate random processes.

C depends on M.

Definition (Perfect Secrecy) An encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is perfectly secret if for every message $m \in \mathcal{M}$ and every ciphertext $c \in \mathcal{C}$ for which $\Pr(\boldsymbol{C} = c) > 0$ we have that $\Pr(\boldsymbol{M} = m \mid \boldsymbol{C} = c) = \Pr(\boldsymbol{M} = m)$.

We also should define some lemma:

Lemma (1) An encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is perfectly secret $\iff$ for every message $m \in \mathcal{M}$ and for every ciphertext $c \in \mathcal{C}$ we have that $\Pr(\boldsymbol{C} = c \mid \boldsymbol{M} = m) = \Pr(\boldsymbol{C} = c)$.

So, basing on conditional probability, you can get the one from the other.

Lemma (2) An encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is perfectly secret $\iff$ for every messages $m_0, m_1 \in \mathcal{M}$ and for every $c \in \mathcal{C}$ we have that $\Pr(\boldsymbol{C} = c \mid \boldsymbol{M} = m_0) = \Pr(\boldsymbol{C} = c \mid \boldsymbol{M} = m_1)$

So, all the messages are equivalent.

2.0.1. Vernam's cipher (aka One-Time Pad)

It is a storical example of a perfectly-secret cipher.

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

$\Pr(\boldsymbol{K} = k) = \frac{1}{2^n}$

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

$\mathsf{Dec}(c, k) = c \oplus k$

This cipher is correct because:

$$\mathsf{Dec}(\mathsf{Enc}(m,k),k) = (m \oplus k) \oplus k = m \oplus (k \oplus k) = m \oplus 0^n = m$$

but there's a limitation on the lengths: messages and keys must have the same length to perform a xor operation. This limitation is a must to have for a perfectly secret encryption scheme. I want to open a bit note about the xor operation which is reversible (and that's because we see it all around this Cryptography class).

Theorem Given a $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ perfectly secret encryption scheme over a message space $\mathcal{M}$ and a key space $\mathcal{K}$. Then $|\mathcal{K}| \geq |\mathcal{M}|$.

Example

We have an example for a private key with an eavesdropper, an adversary A and a scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

```
$\mathrm{PrivK}^{eav}_{A,\Pi}$ :
    $(m_0, m_1) \leftarrow A$
    $k \leftarrow \mathsf{Gen}$
    $b \leftarrow \{0, 1\}$
    $c \leftarrow \mathsf{Enc}(k, m_b)$
    $b^\star \leftarrow A(c)$
    return $\neg(b \oplus b^\star)$
```

The adversary does not guess m_0 or m_1 but b: he can guess c if $b = b^\star$ and that depends on the value 0 or 1 so it is a probability thing.

$$\Pr\left(\mathrm{PrivK}^{eav}_{A,\Pi} = 1\right)$$

This adversary always has a minimal probability of success, so it is useless to require to have that probability $= 0$.

But, it makes sense to have the probability $= \frac{1}{2}$? In this case we say to have a indistinguishable encryptions.

Theorem Π is perfectly secret $\Longleftrightarrow$ Π has indistinguishable encryptions.

2.0.2. Examples

- All of the already seen classic ciphers are not perfectly secure. They have $|\mathcal{K}| < |\mathcal{M}|$. For instance, in Ceaser we have $|\mathcal{K}| = 1$ and, in shift, we have $|\mathcal{K}| = |\Sigma|$ and $|\mathcal{M}| = |\Sigma|^n$. We could define an aversary A such that $\Pr\left(\mathrm{PrivK}^{eav}_{A,\Pi}\right) > \frac{1}{2}$ where Π is any classic of them.

For instance, in the monoalphabetic substitution cipher someone can define an attack in two phases:

1. It produces m_0, m_1 such that $|m_0| = |m_1| = 2$ and $m_0 = \mathsf{aa}$ and $m_1 = \mathsf{ab}$ where $a \neq b$.
2. A looks at c and checks whether $c = a'a'$ or $c = a'b'$ where $a' \neq b'$. In the first case it returns 0, otherwise returns 1.

- Given a $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ where $\mathcal{K} = \mathcal{M} = \mathcal{C} = \{0, 1\}^n$, Gen is the same as in the OTP but $\mathsf{Enc}(m, k) = m \overline{\oplus} k$ with a new operation that is the opposite of xor.

$0 \overline{\oplus} 0 = 1$

$1 \overline{\oplus} 1 = 1$

$0 \overline{\oplus} 1 = 0$

$1 \overline{\oplus} 0 = 0$

We can prove that Π is perfectly secure. The security for the former assures the security for the latter, and we know this uses the security of the OTP (which we know it is secure). So, we use a proof by reduction:

From any A succeeding in breaking Π, we get B succeeding in breaking OTP.

We go backward: starting from the scheme we want to prove secure and go back to a scheme we know it is secure. We reach this negating the thesis and proceeding by contradiction. This strategy is pretty much always used.

How can we define B from A? We can imagine A as a subroutine of B. A produces m_0, m_1 taking in input c' producing in output $b^\star$. m_0, m_1 are forwarded to the environment but the ciphertext c from the environment is different.

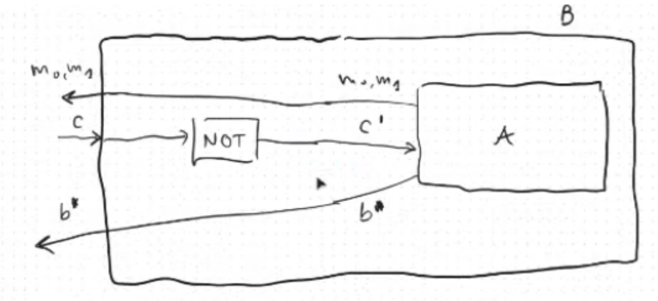


Figure 2: Proof by reduction from A to B

So

$$\Pr\left(\text{Privk}^{eav}_{B,\text{ OTP}} = 1\right) = \Pr\left(\text{Privk}^{eav}_{A,\Pi} = 1\right)$$

- A generalisation of OTP which stretchs a small key to another bigger from an algorithm is the most natural way. An example could be a duplication of each bit: this creates a $2n$ key from a key of length n.

$\text{OTP}^+ = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ where

- Gen is as in OTP.
- $\mathsf{Enc}(m, k) = m \oplus \text{double}(k)$ where $\text{double}(d_1 ... d_n) = d_1 d_1 d_2 d_2 ... d_n d_n$ but this is insecure because $|\mathcal{K}| = |\{0,1\}^n| = 2^n$, $|\mathcal{M}| = \{0,1\}^{2n} = 2^{2n}$ so $|\mathcal{K}| \ll |\mathcal{M}|$.

An example of attack for the thing wrote above is, given

$m_0 = 00$ and $m_1 = 10 \longrightarrow$ for the first phase.

$c = ab$ returns 0 if $d = b$, else 1 $\longrightarrow$ for the second phase.

In opposite of its natural way, this "doubl-ication" is dangerous.

Lemma An encryption scheme is perfectly secure $\Longleftrightarrow$ for every distribution on $\mathcal{M}$ and for $m_0, m_1 \in \mathcal{M}$, it holds that $\Pr(\boldsymbol{C} = c \mid \boldsymbol{M} = m_0) = \Pr(\boldsymbol{C} = c \mid \boldsymbol{M} = m_1) \qquad \forall c \in \mathcal{C}$.

Proof

$(\Longrightarrow)$

$\Pr(\boldsymbol{C} = c \mid \boldsymbol{M} = m_0) = \Pr(\boldsymbol{C} = c)$ because it is the definition of perfect security.

But also $\Pr(\boldsymbol{C} = c \mid \boldsymbol{M} = m_0) = \Pr(\boldsymbol{C} = c \mid \boldsymbol{M} = m_1)$

$(\Longleftarrow)$

By hypothesis we know that $\Pr(\boldsymbol{C} = c \mid \boldsymbol{M} = m_1)$ has the same value for every possible message m_i.

$$\Pr(\boldsymbol{C} = c) = \sum_{m_i \in \mathcal{M}} \Pr(\boldsymbol{C} = c \wedge \boldsymbol{M} = m_i)$$

It is a disjoint because a message can't be two different messages. But also

$$\Pr(\boldsymbol{C} = c) = \sum_{m_i \in \mathcal{M}} \Pr(\boldsymbol{C} = c \mid \boldsymbol{M} = m_i) \cdot \Pr(\boldsymbol{M} = m_i)$$

This comes from the property of $\Pr(A \cap B) = \Pr(A|B) \cdot \Pr(B)$.

Rewriting the formula above using a variable $p = \Pr(\boldsymbol{C} = c \mid \boldsymbol{M} = m_i)$ we can write

$$\begin{aligned}\Pr(\boldsymbol{C} = c) &= \sum_{m_i \in \mathcal{M}} p \cdot \Pr(\boldsymbol{M} = m_i) \\ &= p \cdot \sum_{m_i \in \mathcal{M}} \Pr(\boldsymbol{M} = m_i) \\ &= p \\ &= \Pr(\boldsymbol{C} = c \mid \boldsymbol{M} = m_i)\end{aligned}$$

this since $\sum_{m_i \in \mathcal{M}} \Pr(\boldsymbol{M} = m_i) = 1$.

■

Theorem The OTP is perfectly secure.

Proof

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

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

$\mathsf{Dec}(c, k) = c \oplus k$

Let us fix an arbitrary distribution on $\mathcal{M}$, a message $m \in \mathcal{M}$ and a ciphertext $c \in \mathcal{C}$. We want to prove that $\Pr(\boldsymbol{C} = c \mid \boldsymbol{M} = m)$ is somehow independent on m.

$$\begin{aligned}\Pr(\boldsymbol{C} = c \mid \boldsymbol{M} = m) &= \Pr(\boldsymbol{M} \oplus \boldsymbol{K} = c \mid \boldsymbol{M} = m) \\ &= \Pr(\underbrace{m \oplus k}_{x} = \underbrace{c}_{y}) \\ &= \Pr(m \oplus (m \oplus k) = m \oplus c) \\ &= \Pr((m \oplus m) \oplus \boldsymbol{K} = m \oplus c) \\ &= \Pr(\boldsymbol{K} = m \oplus c) = \tfrac{1}{2^n}\end{aligned}$$

because the bold variables (the random values) can be rewrited.

A good property of the $\oplus$ is:

$y = x \Longleftrightarrow m \oplus y = m \oplus x$

If you fix the value of m then you have a permutation of the all possible strings because the second value can vary. In fact, we're using this property on the second line.

Notably $\frac{1}{2^n}$ does depend on m and so we can conclude that

$\Pr(\boldsymbol{C} = c \mid \boldsymbol{M} = m_0) = \Pr(\boldsymbol{C} = c \mid \boldsymbol{M} = m_1) = \frac{1}{2^n}$.

■

Theorem (Shannon) If an encryption scheme is perfectly secure then $|\mathcal{K}| \geq |\mathcal{M}|$.

Proof

We proceed by contradiction assuming the existence $|\mathcal{K}| < |\mathcal{M}|$. Suppose that $c \in \mathcal{C}$ is such that $\Pr(\boldsymbol{C} = c) > 0$. Let us define

$$\mathcal{M}(c) \overset{\text{def}}{=} \{\hat{m} \mid \hat{m} = \mathsf{Dec}(c, k) \text{ for some } k \in \mathcal{K}\}$$

We have all the possible messages by decryption. The set $\mathcal{M}(c)$ can't be too big. In particular since Dec is deterministic it can't contain more than $|\mathcal{K}|$ messages.

$$|\mathcal{M}(c)| \leq |\mathcal{K}| < |\mathcal{M}|$$

So $\mathcal{M}(c) \subset \mathcal{M}$ and there is a $m' \in \mathcal{M}$ such that $m' \notin \mathcal{M}(c)$.

$\Pr(\boldsymbol{M} = m' \mid \boldsymbol{C} = c) = 0 \neq \Pr(\boldsymbol{M} = m')$

This because $m' \notin \mathcal{M}(c)$. "$\neq$" because there is no assumption whatsoever about the distribution over messages and the encryption scheme is perfectly secure. But, the fact they can't have the same value is a contradiction!

■

3. Private-Key Encryption and Pseudorandomness

We need to work with strings because we need to measure the length of the input. Modern cryptography is all about assumptions.

A concrete approach is a way to define an encryption scheme with a pair made of time of success at most t and a probability at most ε. The t boundary depends on the hardware so it becomes obsolete for new generations.

An asymptotic approach uses a parametric notion of time and error. Everything depends on a security parameter n (it is very convenient). For instance, the length of the key is meant as goal for the security. Efficiency is defined for PPT (Probabilistic Polynomial Time) algorithms but low probability of success is captured by negligible functions (function converges to 0 when security parameter grows, a very low probability). A scheme is secure if every PPT adversary can break the scheme with only negligible probability. Here, we don't depend on underlying hardware.

3.1. PPT

A probabilistic algorithm A is said to be PPT if there's a polynomial p that defines an upper-bound on the computation time of A independently on the outcomes of the performed probabilistic choices.

p bounds the average execution time of A.

This is very similar to the complexity class BPP (see "Modelli concorrenti" class).

You could use Non Deterministic Turing Machine for this kind of algorithms, if you want to.

3.2. Negligible functions

A function $f : \mathbb{N} \to \mathbb{R}$ is said to be negligible $\iff$ for every polynomial $p : \mathbb{N} \to \mathbb{N}$ there exist $N \in \mathbb{N}$ such that for every $n > N$ it holds $f(n) < \frac{1}{p(n)}$.

An example is $n \mapsto 2^{-n}, n \mapsto 2^{-\sqrt{n}}, n \mapsto n^{-\log n}$

Lemma The set of all negligible functions called $\mathcal{NGL}$ is closed for sum, product and product by an arbitrary polynomial.

The class of the lemma above is used for robust security definitions.

If the adversary have no chance to break the class we can't prove anything.

3.3. Limited resources and possible attacks

We assume that $|\mathcal{K}| \gg |\mathcal{M}|$. If this happens an inefficient adversary can be capable to break the scheme.

Assuming $|\mathcal{K}| \ll |\mathcal{M}|$ we could construct two kinds of attacks:

1. in a ciphertext-only context we could decrypt a ciphertext c with all possible keys.
2. we could use a known-plaintext context observing $(m_1, c_1), ..., (m_l, c_l)$ using a key k completely random and then check if $\mathsf{Dec}_k(c_i) = m_i$ for every i. We have a success-probability very very low.

3.4. Assumptions

We already mentioned the impossibility of demonstrating scheme unconditionally.

$$\forall A \in \text{PPT}. \neg\ \mathbf{BRK}(\Pi, A) \iff \neg \exists A \in \text{PPT}.\ \mathbf{BRK}(\Pi, A)$$

We're using De Morgan laws to change the statements in the below forms which use two implications instead of a single *iff*.

The best we can do is an assumption of the same form considering two different schemes. We can do so exploiting the duality of what we actually do.

$$\forall B \in \text{PPT}. \neg\ \mathbf{BRK}(\Xi, B) \Rightarrow \forall A \in \text{PPT}. \neg\ \mathbf{BRK}(\Pi, A)$$

This proof by reduction is observing that formula above holds

$$\Updownarrow$$

$$\exists A \in \text{PPT}.\ \mathbf{BRK}(\Pi, A) \Rightarrow \exists B \in \text{PPT}.\ \mathbf{BRK}(\Xi, B)$$

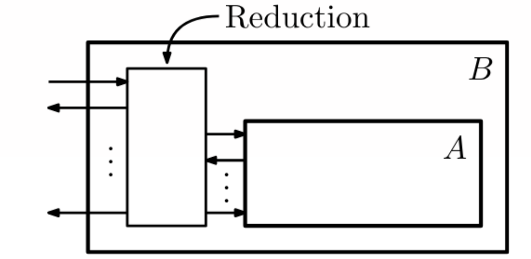


Figure 3: Proof by reduction

3.5. Different definition for "encryption scheme"

We want an efficient algorithm, PPT for sure, which has:

- Gen function takes an input string length 1^n and its output must be long enough, we say $|k| > n$.
- Enc function is probabilistic and produces, of course, different outputs (m, c) from different keys.
- Dec is deterministic.

We say it is correct if $\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = m$.

$\ell(n)$ with n is the security parameter generated by Gen. In this case ℓ is a parameter which grows up more than linear time; we say that encryption scheme has a fixed length.

We want a more clear definition for efficiency!

A security scheme defined as semantic security is, despite its difficulties, mentioned on the book and it is a theorization of the Shannon algorithm. So, another way is the adaption of the notion of experiment about perfect secrecy. We can't be sure which is the length of the message but, for instance, you may know the difference of two ciphertexts from a 10MB and a 4GB files (ciphertexts could have lengths 10MB + some and 4GB + some). If you just pass n the adversay can be capable of finding the length in $\log n$ 'cause its binary nature.

```
$\mathsf{PrivK}^{eav}_{A,\Pi}(n)$:
    $(m_0, m_1) \leftarrow A(1^n)$
    if $|m_0| \neq |m_1|$ then return $0$
    $k \leftarrow \mathsf{Gen}(1^n)$
    $b \leftarrow \{0, 1\}$
    $c \leftarrow \mathsf{Enc}(k, m_b)$
    $b^\star \leftarrow A(c)$
    **return** $\neg(b \oplus b^\star)$
```

Definition (Secure against passive attacks) An encryption scheme Π is said to be secure against passive attacks or secure with respect to $\mathsf{PrivK}^{eav} \iff$ for every PPT adversary A there exist a function $\varepsilon \in \mathcal{NGL}$ such that

$$\Pr\Big(\mathsf{PrivK}^{eav}_{\Pi,A}(n) = 1\Big) = \frac{1}{2} + \varepsilon(n)$$

The probability of success for an adversary can't be much more than $\frac{1}{2}$ cause its binary nature. For example, it can't be 1. You can't pick a negligible function after you know the length.

3.6. Pseudorandom Generator

Starting from the idea of the OTP $k \oplus m = c$ we could use a greater version of the key k such that $G(k) \oplus m = c$.

G function must have some properties: first of all it should work in polynomial time; it can't add randomization because you won't be able to have the same key for decryption; it must have some kind of pseudorandomness not duplicating data as mentioned before.

Definition (Pseudorandom Generator) Given $\ell : \mathbb{N} \to \mathbb{N}$ is a polynomial function called expansion factor. Also, given G a deterministic algorithm that, for every $s \in \{0,1\}^*$ outputs a string $G(s) \in \{0,1\}^{\ell(|s|)}$. We say that G is a pseudorandom generator PRG $\iff$

1. For every $n \in \mathbb{N}$ it holds $\ell(n) > n$.
2. G is polytime.
3. For every PPT algorithm D $\exists \varepsilon \in \mathcal{NGL}$ such that

$$|\Pr(D(s) = 1) - \Pr(D(G(r)) = 1)| \leq \varepsilon(n)$$

where s and r are random of length $\ell(n)$ and n.

It produces something random which is not really is. This function is not always based on polynomial time because it depends on the output of G.

Only a negligible function can return in output another negligible output. For example, given $\ell(n) = 2n$ we have an output of G based on the possible 2^n strings of length $2n$ which is 2^{2n}. But $\frac{2^n}{2^{2n}} = \frac{1}{2^n}$.

We must be sure that a brute-force attack can't be available, and we can be sure about that using a sufficiently large n.

The pseudorandom property can't be proved, so we're assuming to have this kind of generator in our PC, but, as Ugo said, "who knows?".

Definition (PRG-induced scheme) Given a PRG G with expansion factor of ℓ the scheme $\Pi^G = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is

- $\mathsf{Gen} : \mathbb{N} \to \mathbb{N}$ has the same probability for input strings 1^n to have an output string of length n.
- $\mathsf{Enc}(m, k) = G(k) \oplus m$
- $\mathsf{Dec}(c, k) = G(k) \oplus c$

Correctness is defined as $\mathsf{Dec}(\mathsf{Enc}(m, k), k) = (G(k) \oplus m) \oplus G(k) = m$.

Lemma The negligible functions are closed with respect to multiplication by a polynomial, i.e. if $\varepsilon \in \mathcal{NGL}$ and p is any polynomial, then $n \mapsto \varepsilon(n) \cdot p(n)$ is negligible.

Proof

From the hypothesis $\varepsilon \in \mathcal{NGL}$ we know that for every pair of polynomials r, q it holds that

$$\varepsilon(n) < \frac{1}{r(n) \cdot q(n)} \forall n \geq N \qquad (\star)$$

for a certain $N \in \mathbb{N}$.

The product under the bar of the division is also a polynomial.

We now want to test the $\varepsilon \cdot p$ with all possible polynomials.

Proving that $\varepsilon \cdot p$ is negligible, let's consider any polynomial q and prove that for every $n \geq M$ holds $(\varepsilon \cdot p)(n) < \frac{1}{q(n)}$.

We can exploit $(\star)$ and pick $r = p$. This way

$$\begin{aligned}(\varepsilon \cdot p)(n) = \varepsilon(n) \cdot p(n) &< \frac{1}{p(n) \cdot q(n)} \cdot p(n) \\ = \varepsilon(n) &< \frac{1}{q(n)} \forall n \geq N \qquad M = N\end{aligned}$$

M it could be different of what N is from the $(\star)$. So we just keep them separated, not using the usual $\max(N_1, N_2)$.

■

Theorem If G is a PRG $\Rightarrow \Pi^G$ is secure against passive attacks.

Proof

It's done by reduction.

$$\exists A \in \text{PPT}.\ \mathbf{BRK}\big(A, \Pi^G\big) \Rightarrow \exists D \in \text{PPT}.\ \mathbf{BRK}(D_A, G)$$

We're assuming an adversary. Out of any successful adversary A for Π^G a distinguisher D_A (because D depends on A) which uses A as a subroutine.

x works as the pseudorandom input. x is streched here, so we need to compute n first using an inverse of ℓ function.

```
$D_A(x)$ :
    $n \leftarrow \ell^{-1}(|x|)$
    $m_0, m_1 \leftarrow A(1^n)$
    **if** $|m_0| \neq |m_1|$ **then return** $0$
    $b \leftarrow \{0, 1\}$
    $c \leftarrow m_b \oplus x$
    $b^\star \leftarrow A(c)$
    **return** $\neg(b \oplus b^\star)$
```

If A works in polynomial time then D_A works in polynomial time too.

We want to prove that

$$\mathbf{BRK}\left(A, \Pi^G\right) \Rightarrow \mathbf{BRK}(D_A, G)$$

where the first part is

$$\mathsf{PrivK}^{eav}_{A,\Pi^G}(n) \nsim \text{FlipCoin}$$

and the latter is

$$D_A(G(S)) \nsim D_A(r)$$

Breaking a scheme is often a relational property.

Now, we want to prove

$$(1) \qquad \mathsf{PrivK}^{eav}_{A,\Pi^G}(n) = D_A(G(S))$$

and

$$(2) \qquad \text{FlipCoin} = D_A(r)$$

(1) We proceed by

$$\Pr\left(\mathsf{PrivK}^{eav}_{A,\Pi^G}(n) = 1\right) = \Pr(D_A(G(S)) = 1)$$

and to do so we just can see the definitions for $\mathsf{PrivK}^{eav}_{A,\Pi^G}$ and $D_A(G(S))$. The two pseudocodes output the same result.

(2) We proceed looking that

$$\Pr(D_A(r) = 1) \equiv \Pr\left(\mathsf{PrivK}^{eav}_{A,\text{ OTP}} = 1\right) = \frac{1}{2}$$

the latter is

```
$\mathsf{PrivK}^{eav}_{A,\text{ OTP}}(n)$ :
    $k \leftarrow \{0,1\}^{\ell(n)}$
    $m_0, m_1 \leftarrow A(1^n)$
    $b \leftarrow \{0,1\}$
    $c \leftarrow m_b \oplus k$
    $b^\star \leftarrow A(c)$
    $\textbf{return } \neg(b^\star \oplus b)$
```

So we proved that $\Pr(D_A(r) = 1)$ is $\frac{1}{2}$ and so the FlipCoin test is.

■

Example

It is the exercise 2.6 from the Exercise book.

We want to prove that

$$\Pr(A(\mathsf{Enc}(k,m)) = \text{lastbit}(m)) = \frac{1}{2} + \varepsilon(n)$$

where ε is a negligible function. Of course there's a $\frac{1}{2}$ on the result but, that $\varepsilon(n)$ is proved by reduction.

So, the proof is

$$\exists B \in \text{PPT}.\mathbf{BRK}^{lb}(B,\Pi) \Rightarrow \exists A \in \text{PPT}.\mathbf{BRK}^{eav}(A,\Pi)$$

lb is the "last bit". In other words, if an adversary can be capable of break the scheme knowing the last bit, it implies that exists another adversary capable to break the scheme in the usual way.

The adversary runs $\ell(n)$ bits producing randomly m_0, m_1 except for the last one bit.

Let us build the adversary A using B as a subroutine

```
function $A^{\text{FIRST}}(1^n)$ :
    $w \leftarrow \{0,1\}^{\ell(n)-1}$
    return $(w_0, w_1)$
```

```
function $A^{\text{SECOND}}(c)$ :
    return $B(c)$
```

Now, we just want to prove that

$$\Pr(B(\mathsf{Enc}(k,m)) = \text{lastbit}(m)) \neq \text{FlipCoin}$$

$$\Downarrow$$

$$\Pr\Big(\mathsf{PrivK}^{eav}_{A,\Pi}(n) = 1\Big) \neq \text{FlipCoin}$$

but this is the same of what we did in the page before this one because equivalents. So the proving becomes

$$\text{FlipCoin} = \text{FlipCoin}$$

and

$$\Pr(B(\mathsf{Enc}(k,m)) = \text{lastbit}(m)) = \Pr\Big(\mathsf{PrivK}^{eav}_{A,\Pi}(n) = 1\Big)$$

This can be easily proved by examining the pseudocodes of A and B.

Example

This is the exercise 2.7 from the Exercises book.

We have that

$$\Pi \text{ secure}, \Theta \text{ secure} \Rightarrow \Pi\#\Theta \text{ secure}$$

to prove that we could have by reduction that (using De Morgan with two negations) that

$$\exists A.\ \mathbf{BRK}(A, \Pi\#\Theta) \Rightarrow \exists B.\ \mathbf{BRK}(B, \Pi) \lor \exists B.\ \mathbf{BRK}(B, \Theta)$$

but this is not really trivial, we should work a lot on :sad:

But, the other implication

$$\Pi\#\Theta \text{ secure} \Rightarrow \Pi \text{ secure}, \Theta \text{ secure}$$

we proceed by proving

$$\exists B.\ \mathbf{BRK}(B, \Pi) \lor \exists B.\ \mathbf{BRK}(B, \Theta) \Rightarrow \exists A.\ \mathbf{BRK}(A, \Pi\#\Theta)$$

and that's easier because we can split the prove by two cases

$$\exists B.\ \mathbf{BRK}(B, \Pi) \Rightarrow \exists A.\ \mathbf{BRK}(A, \Pi\#\Theta)$$

$$\exists B.\ \mathbf{BRK}(B, \Theta) \Rightarrow \exists A.\ \mathbf{BRK}(A, \Pi\#\Theta)$$

For example, the first one can be proved by building an adversary for $\Pi\#\Theta$ from an adversary for Π.

```
function $B^{\text{FIRST}}(1^n)$ :
    $\langle m_0^\Pi, m_1^\Pi \rangle \leftarrow A(1^n)$
    return $\langle m_0^\Pi \cdot m, m_1^\Pi \cdot m \rangle$
```

where m is any fixed message.

```
function $B^{\text{SECOND}}(c)$ :
    $\langle c^\Pi, c^\Theta \rangle \leftarrow c$
    return $A^{\text{SECOND}}(c^\Pi)$
```

we strip the last part and fill the rest. We can't stop here sadly.

$$\Pr\Big(\mathsf{PrivK}_{A,\Pi}^{eav}(n) = 1\Big) = \frac{1}{2} + \eta(n)$$

$$\Downarrow$$

$$\Pr\Big(\mathsf{PrivK}_{B,\Pi\#\Theta}^{eav}(n) = 1\Big) = \frac{1}{2} + \xi(n)$$

where η, ξ are not negligible.

3.6.1. Variable-Length messages

First of all we need to define the algorithm below.

Definition (Variable Output-Length Generator) A deterministic polytime algorithm G is said to be Variable Output-Length Pseudorandom Generator if from a seed $s \in \{0,1\}^n$ and a string in the form 1^ℓ outputs a binary string $G(s, 1^\ell)$ such that

1. if $\ell < \ell'$ the string $G(s, 1^{\ell})$ is a prefix of $G(s, 1^{\ell'})$.
2. for any polynomial $p : \mathbb{N} \mapsto \mathbb{N}$ the algorithm G_p defined by setting $G_p(s) = G\big(s, 1^{p(|s|)}\big)$ is a random generator.

The output can be much longer pumping the second parameter. This is not limited by an expansion factor. This produces a stream of bits.

Now, given a variable output-length pseudorandom generator G we can generalize Π^G as

- $\mathsf{Enc}(m, k) = G\big(k, 1^{|m|}\big) \oplus m$
- $\mathsf{Dec}(c, k) = G\big(k, 1^{|c|}\big) \oplus c$

Lemma For every G it's possible to construct a variable output-length generator H from G.

RC4 from Linux (https://en.wikipedia.org/wiki/RC4) works pretty much like that. They're not encryption schemes and can't be proved that they satisfy these axioms.

Despite what the book says, they can be used for a lot of things other than just ciphers.

3.6.2. Multiple Encryptions

The ciphertexts seen until now are just a string c, not a splitted string (or at the least it is what we assumed) but this is restrictive.

An adversary which outputs two vectors over two strings of messages is defined by $\mathsf{PrivK}^{\text{mult}}$.

The key in OTP is supposed to be used only once, so it is not secure here.

So, the adversary produces $\boldsymbol{m_0} = (m_0^1, ..., m_0^t)$ and $\boldsymbol{m_1} = (m_1^1, ..., m_1^t)$ so the cipher becomes $\boldsymbol{c} = (c_1, ..., c_t)$.

$$\Pr\Big(\mathsf{PrivK}^{\text{mult}}_{\Pi,A}(n) = 1\Big) = \frac{1}{2} + \varepsilon(n)$$

Lemma Π^G is not secure with respect to $\mathsf{PrivK}^{\text{mult}}$, not even if G is pseudorandom.

Proof

Let us define an adversary A against Π^G, showing that

$$\Pr\Big(\mathsf{PrivK}^{\text{mult}}_{A,\Pi^G}(n) = 1\Big) = \frac{1}{2} + \eta(n)$$

with η not negligible.

```
function $A^{\text{FIRST}}(1^n)$ :
    return $\langle\big(0^{\ell(n)}, 0^{\ell(n)}\big), \big(0^{\ell(n)}, 1^{\ell(n)}\big)\rangle$
```

```
function $A^{\text{SECOND}}(\boldsymbol{c})$ :
    $\langle c_1, c_2\rangle \leftarrow \boldsymbol{c}$
    if $c_1 = c_2$ then return 0
    else return 1
```

we can explicitly analyse the probability

$$
\begin{aligned}
\Pr\Big(\mathsf{PrivK}^{\text{mult}}_{A,\Pi^G} = 1\Big) ={}& \\
&= \frac{1}{2}\Pr\Big(\mathsf{PrivK}^{\text{mult}}_{A,\Pi^G}(n) = 1 \mid b = 0\Big) + \\
&\frac{1}{2}\Pr\Big(\mathsf{PrivK}^{\text{mult}}_{A,\Pi^G}(n) = 1 \mid b = 1\Big) = \\
&= \frac{1}{2}\cdot 1 + \frac{1}{2}\cdot 1 = \frac{1}{2} + \frac{1}{2} = 1
\end{aligned}
$$

■

Theorem If Enc is deterministic, then the scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ can't be secure with respect to $\mathsf{PrivK}^{\text{mult}}$.

So, the stream ciphers are not useless in the context of multiple encryptions but we need Enc with an internal state.

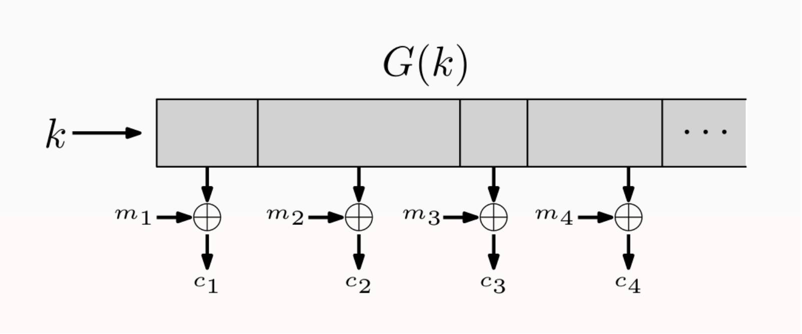


Figure 4: Synchronized mode

But this has a kind of state.

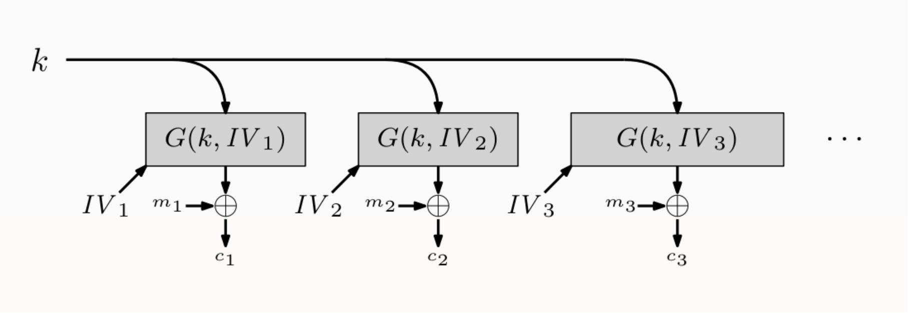


Figure 5: Unsynchronized mode

This last one uses an **IV**, the Initialization Vector. It introduces some confusion, like a randomness.

3.6.3. Security against CPA attacks

We have the assumption of the oracle for $\mathsf{Enc}_k(\cdot)$ which, once called for a message m, it'll returns a $c = \mathsf{Enc}_k(m)$. It's just some constraint of how many times the adversary can ask to the oracle in a time.

Definition An encryption scheme Π is secure against CPA attack (it's an active attack) $\iff$ for every adversary A there exist a negligible function ε such that

$$\Pr\Big(\mathsf{PrivK}^{\text{CPA}}_{A,\Pi}(n) = 1\Big) \leq \frac{1}{2} + \varepsilon(n)$$

Since this kind of active attacks are more efficacy we need some lemma.

Lemma (1) Any scheme Π that is secure with respect to $\mathsf{PrivK}^{\mathrm{CPA}}$ is secure with respect to PrivK^{eav}.

Lemma (2) Any scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ that is secure with respect to $\mathsf{PrivK}^{\mathrm{CPA}}$ must be such that Enc is probabilistic.

Theorem Every encryption scheme that is CPA-secure is secure even in case of multiple encodings.

3.7. Pseudorandom Function

The output of a pseudorandom generator is a stream but a pseudorandom function returns a function.

- It takes two strings as input. E.g. $\{0,1\}^\star \times \{0,1\}^\star \to \{0,1\}^\star$.
- A binary partial function F is length-preserving $\iff F(k,x)$ is defined $\iff |k| = |x|$ and in that case $|F(k,x)| = |x|$. It's only defined when the two strings are the same length.
- Given a length-preserving binary partial function F, we denote by F_k the function $\{0,1\}^{|k|} \mapsto \{0,1\}^{|k|}$ defined in the natural way. This it the currying for function programming.
- A binary partial function is efficient $\iff$ there exist a polytime algorithm that computes it.
- We consider the space of functions $\{0,1\}^n \mapsto \{0,1\}^n$.

  This space is finite and has cardinality $2^{n \cdot 2^n}$, because any of its function can be seen as a table of binary values with 2^n rows and n columns.

It therefore makes sense to consider uniform distribution on such a space, which assigns probability $\frac{1}{2^{n \cdot 2^n}}$ to every function.

Definition Given a binary partial function F, which is length-preserving and efficient, we say that F is a pseudorandom function (PRF) $\iff$ for every distinguisher D that is PPT there exist a negligible function ε such that

$$|\Pr\big(D^{F_k(\cdot)}(1^n) = 1\big) - \Pr\big(D^{f(\cdot)}(1^n) = 1\big)| \leq \varepsilon(n)$$

D is a little bit different than other functions, such as generators. It's linked to two different functions here.

- k is chosen among all strings of length n randomly.
- $f(\cdot)$ is chosen among all functions from $\{0,1\}^n$ to $\{0,1\}^n$ randomly. It's very hard to predict the output unless you've already called the function with the same input.

So it calls the function without knowing F_k.

Considering $F(k,x) = x$, we do not have F pseudorandom because a D could query the oracle on 0^n, output 1 if the result of the query is 1 and 0 otherwise.

$\Pr\big(D^{F_k(\cdot)}(1^n) = 1\big) = 1$ this because $F(k,x)$ returns x.

$\Pr\big(D^{f(\cdot)}(1^n) = 1\big) = \frac{1}{2^n}$

thus

$$|\Pr\big(D^{F_k(\cdot)}(1^n) = 1\big) - \Pr\big(D^{f(\cdot)}(1^n) = 1\big)| \leq 1 - \frac{1}{2^n}$$

and that $1 - \frac{1}{2^n}$ is not negligible.

As with pseudorandom generators, the existence of pseudorandom functions is not known in an absolute sense. From a pseudorandom generator we can construct a pseudorandom function and viceversa. They're called block ciphers.

Definition (PRF-Induced Scheme) Given a pseudorandom function F, the scheme $\Pi^F = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is defined as follows:

- The algorithm Gen on input 1^n outputs each string of length n with same probability, i.e. $\frac{1}{2n}$.
- $\mathsf{Enc}(m, k)$ is defined as the (or binary encoding) pair $\langle r, s \rangle$, where r is a random string $|k|$ bits long and $s = F_k(r) \oplus m$. That $\oplus$ operation looks random unless some bad events those eventually could happen not really frequently.
- $\mathsf{Dec}(c, k)$ returns $F_k(r) \oplus s$. So we have $F_k(r) \oplus (F_k(r) \oplus m) = 0 \oplus m = m$.

So, Π^F is a correct encryption scheme. The key k is unknown ofc. This scheme is commonly used on Internet.

Theorem F is a PRF $\Rightarrow \Pi^F$ is secure against CPA attacks.

Proof

It's divided in two parts.

1. We study a $\tilde{\Pi} = \left(\widetilde{\mathsf{Gen}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Dec}}\right)$ which is an idealized version of Π^F. Here,
   - $\widetilde{\mathsf{Gen}}$, rather than generating a n-bit string, it generates a random function from $\{0,1\}^n$ to itself, namely it fills a truth table of the Figure 6.

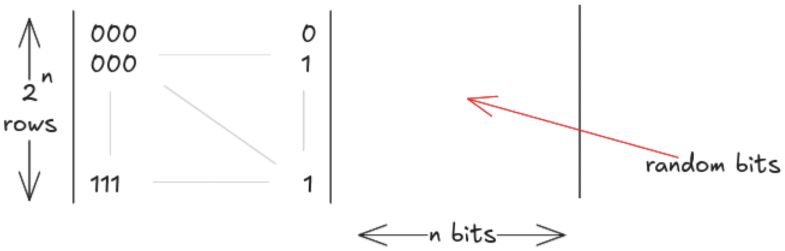


Figure 6: Truth table

   This means $\widetilde{\mathsf{Gen}}$ is not efficiently computable but this is not a problem because $\tilde{\Pi}$ is just an idealized scheme, it won't be used in pratice.
   - $\widetilde{\mathsf{Enc}}$ is defined similarly to

```
Enc^F(m):
    $r \leftarrow \{0,1\}^n$
    return $\langle r, f(r) \oplus m \rangle$
```

   where $f(r)$ is the key for now.
   - $\widetilde{\mathsf{Dec}}$ is defined as above for $\widetilde{\mathsf{Enc}}$.

How to prove that $\tilde{\Pi}$ is CPA-secure without any assumptions? We just have to look at the interaction between any adversary A and $\tilde{\Pi}$ in the experiment $\mathsf{PrivK}^{\text{CPA}}$. We should now see about $\tilde{\Pi}$. It can in particular query the encryption oracle and get from it some results, in the following form

$$\langle d_1, \langle r_1, s_1\rangle\rangle, \langle d_2, \langle r_2, s_2\rangle\rangle, ..., \langle d_\ell, \langle r_\ell, s_\ell\rangle\rangle$$

where

d_i is the i-th message.

r_i is the i-th random value generated internally by $\widetilde{\mathsf{Enc}}$. It's a completely random row in the truth table of the Figure 6.

$s_i = f(r_i) \oplus m_i$.

The adversary, moreover, also receives the "challenge ciphertext", namely $\langle r, s\rangle$ such that $s = f(r) \oplus m_b$ where b is random.

The adversary has a "bella botta di culo" when r is equals to one of the r_i. The reasoning we are going to do is based on the probabilist event $r = r_i$, we call `Repeat`.

- If `Repeat` holds, then they could easily determine $f(r) = f(r_i)$ and thus m_b because they already know b.
- If `Repeat` does not hold, then A cannot guess anything about b, because $f(r)$ would be a genuinely random value about which A knows nothing

Now,

$$\begin{aligned}\Pr\Big(\mathsf{PrivK}^{\text{CPA}}_{A,\tilde{\Pi}}(n) = 1\Big) &= \underbrace{\Pr\Big(\mathsf{PrivK}^{\text{CPA}}_{A,\tilde{\Pi}}(n) = 1 \mid \texttt{Repeat}\Big)}_{=1} \cdot \Pr(\texttt{Repeat}) + \\ &+ \underbrace{\Pr\Big(\mathsf{PrivK}^{\text{CPA}}_{A,\tilde{\Pi}}(n) = 1 \mid \neg\ \texttt{Repeat}\Big)}_{=1/2} \cdot \underbrace{\Pr(\neg\ \texttt{Repeat})}_{=1} \leq \\ &\leq 1 \cdot \Pr(\texttt{Repeat}) + \frac{1}{2} \cdot 1\end{aligned}$$

We can do that because $\Pr(A) = \Pr(B) \cdot \Pr(A|B) + \Pr(\neg B) \cdot \Pr(A|\neg B)$.

$\Pr(\texttt{Repeat})$ can't be too big because $\ell \leq q(n)$ where q is polynomial as a consequence $\Pr(\texttt{Repeat})$ is upper-bounded by $\frac{q(n)}{2^n}$ thanks to the fact that $\Pr(r = r_i) = \frac{1}{2^n}$ if r is fixed string, and exploiting union bounds $\Pr(A \cup B) \leq \Pr(A) + \Pr(B)$.

So,

$$\begin{aligned}\Pr\Big(\mathsf{PrivK}^{\text{CPA}}_{A,\tilde{\Pi}}(n) = 1\Big) &\leq \frac{1}{2} + \frac{q(n)}{2^n} \\ &= \frac{1}{2} + \varepsilon(n)\end{aligned}$$

where $\frac{q(n)}{2^n}$ is negligible.

2. The second part is a properly reduction. In particular we assume A breaks Π^F namely that $\Pr\Big(\mathsf{PrivK}^{\text{CPA}}_{A,\Pi^F}(n)\Big) = \frac{1}{2} + \eta(n)$ where η is not negligible, and we build from A a distinguisher D_A for F. This function D_A has access to an oracle for either $F_k(\cdot)$ or $f(\cdot)$ named H.

function $D_A(1^n)$:

- we call $A(1^n)$ and wait until it produces m_0, m_1.

  If in the meantime A calls the oracle for $\mathsf{Enc}_k(\cdot)$ on a value m, we prooced by

  - creating a random value r.

  - feeding H with r, obtaing s.
  - we compute $s \oplus m = t$.
  - we return $\langle r, t \rangle$.
- we draw b at random.
- we compute the encryption of m_b by using H thus simulating Enc_k.
- we feed the the obtained ciphertext to A which returns $b^\star$. If A queries $\mathsf{Enc}_k(\cdot)$ we have to proceed as before.
- we return $\neg(b \oplus b^\star)$.

$$\mathsf{PrivK}^{\text{CPA}}_{A,\Pi^F}(n) \neq \text{FlipCoin}$$

$$\Downarrow$$

$$D_A^{F_k(\cdot)}(1^n) \neq D_A^{f(\cdot)}(1^n)$$

The left equivalence is a consequence of out way of definition for D_A.

For the right equivalence, we proved, in the step 1, $\Pr\Big(\mathsf{PrivK}^{\text{CPA}}_{A,\tilde{\Pi}}(n) = 1\Big) = \text{FlipCoin} = \frac{1}{2}$ and this is an equivalence because equals to $D_A^{f(\cdot)}(1^n)$.

In other words

$$\begin{aligned}|\Pr(D_A^{F_k(\cdot)}(1^n) = 1) - \Pr(D_A^{f(\cdot)}(1^n) = 1)| = \\ &= |\Pr(\mathsf{PrivK}^{\text{CPA}}_{A,\Pi^F}(n) = 1) - \Pr(\mathsf{PrivK}^{\text{CPA}}_{A,\tilde{\Pi}}(n) = 1)| = \\ &= |\ \frac{1}{2} + \eta(n) - \frac{1}{2}\ | = \eta(n)\end{aligned}$$

and η is not negligible.

■

For what concern what we've seen at Section 3.6.1, the Π^F does not resolve the issue, having some limits. But, any CPA-secure cipher Π can be generalized for messages of length n to a cipher $\Pi^\star$ which cuts the messages. Now we have messages $n\ell(n)$ where ℓ is polynomial. It works because each step is randomized.

$\mathsf{Enc}^\star\left(k, m_0 \parallel ... \parallel m_{\ell(n)}\right) = \mathsf{Enc}(k, m_0) \parallel ... \parallel \mathsf{Enc}\left(k, m_{\ell(n)}\right)$

Theorem If Π is CPA-secure, then $\Pi^\star$ is also CPA-secure.

3.8. Pseudorandom Permutations

A stronger version uses permutations. An example, for the example above section, you do not decrypt calling the inverse of F_k, but we could do so. This because a permutation of a set X is a bijective function $X \mapsto X$. The number of permutations for a set $\{0,1\}^n$ is $(2^n)!$.

This permutation is effienctly computable.

$$|\Pr\left(D^{F_k(\cdot), F_k^{-1}(\cdot)}(1^n) = 1\right) - \Pr\left(D^{f(\cdot), f^{-1}(\cdot)}(1^n) = 1\right)| \leq \varepsilon(n)$$

The distinguisher D has the oracle and its inverse. Most of all block ciphers (like AES or DES) look like that, using the notion of strongly pseudorandom permutation.

3.9. Modes

The mode operation proceeds splitting the mesage into chunks (said blocks) and you call the pseudorandom permutation function once (or more??) for each. So, for a message m we have $m_1, ...m_\ell$ each for a multiple of $|m|$ called $|k|$.

- **ECB** mode is not secure 'cause its deterministic F_k.
- **CBC** uses an IV so it could be better, and F_k works as permutator.
- **OFB** works like CBC but it can be parallelized because you calculate $m_i \oplus F_k$ but F_k has IV as parameter, but you won't need to precalculate all $m_1, ..., m_{i-1}$ first.
- **CTR** uses a counter instead of a string IV. So, for $m_i \oplus F_k$ the F_k has $C + i$ as input. This is truly parallelized.

4. Authentication

It is always a scenario used in insecure channels but the adversary A works in a direct-mode (in opposite of its use in secrecy).

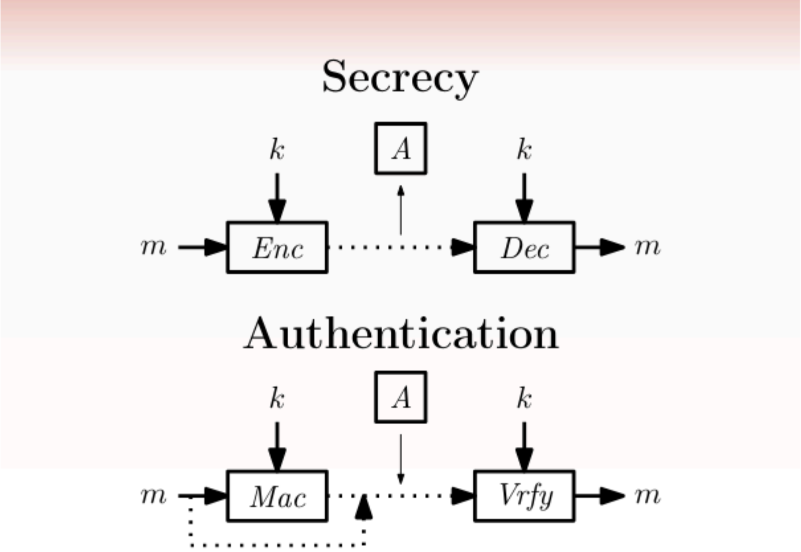


Figure 7: Secrecy vs Auth

MAC is all the scheme, but Mac is used as algorithm.

Authenticity is also named integrity. Solve the issue of secrecy doesn't automatically solve the authentication issue too.

A MAC $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ is correct when $\mathsf{Vrfy}(k, m, \mathsf{Mac}(k, m)) = 1$. We want to prove authenticity, or better, we want to verify the tag (= the Mac output). An attacker should not be able to forge tags of message m without knowing k because the verification by the user would be OK and that's not good at all.

Let's create an adversary which uses an oracle $\mathcal{O} = \mathsf{Mac}_k(\cdot)$. Instead, a pair (m, t) obtained through access to the oracle is not considered a correct forge and also the adversary should be able to do it as PPT algorithm. This kind of attack is not considered successfully.

The security for this attack is defined by the pseudocode below.

```
$\mathsf{MacForge}_{A,\Pi}(n)$ :
    $k \leftarrow \mathsf{Gen}(1^n)$        # *shared key*
    $(m, t) \leftarrow A(1^n, \mathsf{Mac}_k(\cdot))$        # *forge of A*
    $\mathbb{Q} \leftarrow \{s \mid A \text{ queries Mac}_k(\cdot) \text{ on } m\}$        # *set of messages given by A to the oracle with a valid tag*
    **return** $(m \notin \mathbb{Q} \land \mathsf{Vrfy}(k, m, t) = 1)$
```

This not always returns 1 because by definition of correctness we want to return 1 only in some cases. $\mathbb{Q}$ computes all the queries of the attacker wants to perform, collecting all the messages: the result depends on this set.

It's one-shot attack, we don't split it in two phases.

Definition A MAC Π is secure $\iff$ for all PPT adversary A $\exists \varepsilon \in \mathcal{NGL}$ such that $\Pr\big(\mathsf{MacForge}_{A,\Pi}(n) = 1\big) = \varepsilon(n)$.

We do not have a $\frac{1}{2} + \varepsilon(n)$ because, given a boolean output, the probability to be $\frac{1}{2}$ is very high.

An adversary is interested in forging meaningful messages but replay attacks can be possible here. The adversary just could send the pair (m, t) multiple times over the channel, the first time by a legitimate user and the others by the adversary. The replay attack is another level of abstraction.

4.1. Secure MAC

How to construct a secure MAC? We want to achieve security by pseudo randomness as the same as the things seen until now.

Definition (PRF induced MAC) Given a PRF F, the MAC $\Pi^F = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ is defined as:

- Gen for input 1^n outputs strings long n with the probability of 2^{-n}.
- $\mathsf{Mac}(k, m) = F_k(m)$. *Really easy.*
- $\mathsf{Vrfy}(k, m, t) = (F_k(m) \overset{?}{=} t)$. Basically you recompute the $\mathsf{Mac}(k, m)$ and check the output with the supposed tag t.

Of course, if you change a single bit of m you'll receive a completely different tag. **It must be indistinguishable from a random function.** It's not to be considered collision-resistant.

Theorem F is a PRF $\Rightarrow$ MAC Π^F is secure.

Proof

This requires building an idealized MAC $\tilde{\Pi}$, which is a variation of Π^F in which $\widetilde{\mathsf{Gen}}$ instead of sampling k uniformly at random, generates a function from $\{0,1\}^n$ to itself randomly. Of course, then $\widetilde{\mathsf{Mac}}(m, f) = f(m)$ we can prove that $\tilde{\Pi}$ is secure, because guessing the value of $\mathsf{Mac}(k, m)$ without knowing anything about $f(m)$ is simply impossible (unless with negligible probability).

We have somehow to "compare" Π^F and $\tilde{\Pi}$ and prove that they do not behave so differently, unless F is not pseudorandom.

As usual, then, we build a distinguisher for F using an adversary A for Π^F as a subroutine and following the idea that D_A should call A in such a way that has to pretend A in running as part of $\mathsf{MacForge}_{A,\Pi^F}$. In doing so, we get these two equations:

$$\Pr(D_A^{F_k(\cdot)}(1^n) = 1) = \Pr\big(\mathsf{MacForge}_{A,\Pi^F}(n) = 1\big) \qquad (\star)$$

$$\Pr(D_A^{f(\cdot)}(1^n) = 1) = \Pr(\mathsf{MacForge}_{A,\tilde{\Pi}}(n) = 1) = \varepsilon(n) \qquad (\star\star)$$

If, now Π^F is not secure, namely

$$\Pr\big(\mathsf{MacForge}_{A,\Pi^F}(n) = 1\big) = \eta(n) \qquad (\eta \text{ is not negligible})$$

then we would have that

$$|\underbrace{\Pr\Big(D_A^{f_k(\cdot)}(1^n) = 1\Big)}_{(\star\star)} - \underbrace{\Pr\Big(D_A^{F(\cdot)}(1^n) = 1\Big)}_{(\star)+\text{ hypothesis}}| = |\varepsilon(n) - \eta(n)|$$

but $\varepsilon \in \mathcal{NGL}$ and $\eta \notin \mathcal{NGL}$. The result above can't be $\in \mathcal{NGL}$, namely F cannot be pseudorandom, contradicting the hypothesis.

■

Since MAC is mostly equals to PRF, we can apply the MAC to only messages with length $\leq |k|$ and all the properties of PRF hold.

4.2. Handling Variable-Length messages

We have the same issue of the length limit: the message must have the same length of the key.

Given a $m = m_1 \parallel \cdots \parallel m_n$ we want to concatenate to a single tag.

There are some cases:

1. We could auth $\bigoplus_{i=1}^{n} m_i$ but an adversary would easily succeed in forging $p = p_1 \parallel \cdots \parallel p_n$ because $\bigoplus_{i=1}^{n} m_i = \bigoplus_{i=1}^{n} p_i$ flipping two bits from both sides.
2. We could auth each block m_i separately and calculate $\mathsf{Mac}(k, m) = \bigoplus_{i=1}^{n} \mathsf{Mac}(k, m_i)$. But an adversary would easily succeed in forging $p = m_{\pi(1)} \parallel \ldots \parallel m_{\pi(n)}$ where π is a permutation.
3. We could auth each block m_i separately with the sequence number i, $\mathsf{Mac}(k, m) = \bigoplus_{i=1}^{n} \mathsf{Mac}(k, m_i \parallel i)$. But an adversary would succeed anyway.

A solution is to construct a MAC $\Pi^\star = (\mathsf{Gen}, \mathsf{Mac}^\star, \mathsf{Vrfy}^\star)$ with

```
$\mathsf{Mac}^\star(k, m)$:
  $m_1 \parallel \cdots \parallel m_d \leftarrow m$        # Such that $|m_i| = \frac{n}{4}$
  $\ell \leftarrow |m|$
  $r \leftarrow \{0,1\}^{\frac{n}{4}}$
  for $i \leftarrow 1$ to $d$ do
    $t_i \leftarrow \mathsf{Mac}(k, r \parallel \ell \parallel i \parallel m_i)$
  return $(r, t_1, \ldots, t_d)$
```

```
$\mathsf{Vrfy}^\star(k, m, (r, t_1, \ldots, t_d))$:
  $m_1 \parallel \cdots \parallel m_d \leftarrow m$        # Such that $|m_i| = \frac{n}{4}$
  $\ell \leftarrow |m|$
  for $i \leftarrow 1$ to $d$ do
    if $\mathsf{Vrfy}(k, r \parallel \ell \parallel i \parallel m_i) = 0$ then
      return $0$
  return $1$
```

r parameter is not confidential.

You authenticate a random value fixed for each chunk and then authenticate ℓ which is the length of the message. Indeed you need to return also the random value r. All chucks have the same length. The tag is even longer than the message.

Theorem Π secure $\Rightarrow \Pi^\star$ secure.

4.2.1. CBC-MAC

Using CBC we would avoid the longer tag because we'd have a new t_i on every iteration.

This algorithm is truly deterministic not adding any randomness. ℓ returns how many chunks we have.

```
$\mathsf{Mac}^{\mathrm{CBC}}(k, m)$:
    $\ell \leftarrow \ell(|k|)$
    $m_1 \parallel \cdots \parallel m_\ell \leftarrow m$        # Such that $|m_i| = |k|$
    $t_0 \leftarrow 0^n$
    for $i \leftarrow 1$ to $\ell$ do
        $t_i \leftarrow F_k(t_{i-1} \oplus m_i)$
    return $t_\ell$
```

```
$\mathsf{Vrfy}^{\mathrm{CBC}}(k, m, t)$:
    if $\ell(|k|) \cdot |k| \neq |m|$ then
        return 0
    return $t \overset{?}{=} \mathsf{Mac}^{\mathrm{CBC}}(k, m)$
```

Theorem ℓ polynomial and F is PRF $\Rightarrow \Pi^{\mathrm{CBC}}$ is a secure MAC.

4.3. Hash functions

A function which compresses an input string to another much smaller.

$$H : \{0,1\}^\star \times \{0,1\}^\star \mapsto \{0,1\}^\star$$

First param is a key s; the second param is string x.

If $H(x) = H(y)$ with $x \neq y$ we have a collision.

Definition An hash function is a pair of PPT algorithms (Gen, H) such that:

- Gen takes an input 1^n and returns a string s.
- there exist a polynomial ℓ such that $H(s, x)$ returns a string $|\ell(n)|$.

If there exist a polynomial p such that $p(n) > \ell(n)$, for every n, $H(s, x)$ is defined only when $|x| = p(n)$ (and s is the parameter implicit in s), then H is called a fixed-length hash function.

As usual, we denote H_s the function that, for input x, returns $H(s, x)$ [a.k.a. $H_s(x) = H(s, x)$].

4.3.1. Collision resistant

```
$\mathsf{HashColl}_{A,\Pi}(n)$:
    $s \leftarrow \mathsf{Gen}(1^n)$
    $(x, y) \leftarrow A(s)$
    return $(x \neq y) \wedge (H(x) = H(y))$
```

The key is public but the attacker tries to brute force any two messages to find a collision. An use of an oracle $\mathcal{O}$ is really useless.

Seed controls the complexity of this collision.

Definition An hash function $\Pi = (\mathsf{Gen}, H)$ is collision-resistant $\iff$ for every adversary PPT A there exist a negligible function ε such that $\Pr\big(\mathsf{HashColl}_{A,\Pi}(n) = 1\big) \leq \varepsilon(n)$.

You ask to the adversary to find collision with some constraints. Some of them:

- **Second pre-image resistance**: give the adversary the seed s and x and then must find $y \neq x$ such that $H_s(x) = H_s(y)$.
- **Pre-image resistance**: given s and $z = H_s(x)$ it must be impossible to find y such that $z = H_s(y)$. But here the adversary does not know x, so it could be seen like an inversion function (but that is not).

The task is to find any collision, not all the possible $x \neq y$ which cause a collision.

4.3.2. Birthday attack

The length of the seed should be fixed. Here we want to brute force an hash function and check the attack complexity. We have a $H_s : \{0,1\}^\star \mapsto \{0,1\}^\ell$. It's like the birthday paradox: bigger the group → easier to find a collision.

Theorem (Birthday theorem) Given q uniformly chosen random values in a finite set of cardinality N, the probability that two of them are identical is $\Theta\left(\frac{q^2}{N}\right)$.

The curve starts from 0 and grows depending on N. It is still exponential but smoothed by N.

If we assume that the behaviour of H_s is as close as possible to that of a random function (i.e. if we are in the worst case), we can then conclude that a birthday attack where $q = \Theta(2^{\frac{\ell}{2}})$ will have probability of success

$$\Theta\left(\frac{\left(2^{\frac{\ell}{2}}\right)^2}{2^\ell}\right) = \Theta\left(\frac{2^\ell}{2^\ell}\right) = \Theta(1)$$

4.3.3. Merkle-Damgård transform

Suppose that (Gen, H) is an hash function for messages of length $p(n) = 2n$ with output n long and construct from it (Gen, H^{MD}) as

```
$H^{MD}(s, x)$ :
    $B \leftarrow \lceil(|x|)/n\rceil$
    $x_1 \| \dots \| x_B \leftarrow x$        # such that $|x_i| = n$
    $x_{B+1} \leftarrow |x|$
    $z_0 \leftarrow 0^n$
    for $i \leftarrow 1$ to $B + 1$ do
        $z_i \leftarrow H(s, z_{i-1} \| x_i)$
    return $z_{B+1}$
```

The first chuck is compressed many times, like a cascade scheme. z is like an IV.

Theorem If (Gen, H) is collision-resistant, then so is (Gen, H^{MD}).

The proof can't be done by only a simple reduction. You have to go back in a polynomial way.

4.3.4. Use in MAC

We can't put them together in a MAC. In fact, given a MAC $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ and an hash function (Gen', H) we define MAC $\Pi^H = (\mathsf{Gen}^H, \mathsf{Mac}^H, \mathsf{Vrfy}^H)$ as:

- Gen^H on input 1^n and output $\langle s, k\rangle$ where s is the result of $\mathsf{Gen}'(1^n)$ and k is the result of $\mathsf{Gen}(1^n)$. You already use the hash function to compress the message.

- $\mathsf{Mac}^H(\langle s,k\rangle,m)$ outputs $\mathsf{Mac}_k(H_s(m))$.
- $\mathsf{Vrfy}(\langle s,k\rangle,m,t)=1 \iff \mathsf{Mac}^H(\langle s,k\rangle,m)=t$.

Theorem If Π is a secure MAC and (Gen',H) is collision-resistant, then Π^H is secure.

Theorem Π secure MAC and H is collision-resistant hash function $\Rightarrow$ Π^H is secure itself as a MAC.

Proof

Recalling $\Pi^H=(\mathsf{Gen}^H,\mathsf{Mac}^H,\mathsf{Vrfy}^H)$ where $\mathsf{Mac}^H(\langle s,k\rangle,m)=\mathsf{Mac}(k,H_s(m))$.

Before doing the actual reduction, left us analyse the situation from the point of view of an adversary A for Π^H. A can query the oracle for $\mathsf{Mac}_K^H(\cdot)$ and, at some point, outputs $\langle m^\star,t^\star\rangle$.

Let us define the following probabilistic event.

$$\text{coll}_A = \text{"}H_s(m^\star)=H_s(m) \text{ for some } \; m\neq m^\star, m\in\mathbb{Q}\text{"}$$

We can now do some easy probabilistic reasoning:

$$\begin{aligned}\Pr\big(\mathsf{MacForge}_{A,\Pi^H}(n)=1\big) &= \Pr\big(\mathsf{MacForge}_{A,\Pi^H}(n)=1\wedge \text{coll}_A\big)+\\ &\quad+\Pr\big(\mathsf{MacForge}_{A,\Pi^H}(n)=1\wedge \neg\text{coll}_A\big)\le\\ &\le \quad \underbrace{\Pr(\text{coll}_A)}_{\substack{\text{Prove it is negligible}\\ \text{by reduction and exploting}\\ \text{the collision resistant of } H}} \quad + \underbrace{\Pr\big(\mathsf{MacForge}_{A,\Pi^H}(n)=1\wedge \neg\text{coll}_A\big)}_{\substack{\text{Prove that is neglibible by another reduction}\\ \text{and exploting the security of } \Pi}}\end{aligned}$$

(i) In the first reduction we build an adversary C for the hash function using A as a subroutine. Our objective is to prove that

$$\Pr\big(\text{HasColl}_{C_A,H}(n)=1\big)=\Pr(\text{coll}_A)$$

where C_A is defined as:

- First, it produces a key $\langle s,k\rangle$ by calling Gen^H.
- Then, it calls A on 1^n and waits until A produces a result.
- Whenever A queries the oracle for Mac^H on m, C proceeds as follows:
  - It first calls H_s on m and Mac_k on the obtained result.
  - It keeps track of the message m in an internal “database”, call it $\mathbb{D}$, also keeping track of $H_s(m)$.
  - Finally, it forwards the result to A.
- After performing some queries, A finally produces a pair $\langle m^\star,t^\star\rangle$.
- We throw away $t^\star$ and we compute $H_s(m^\star)$ checking in $\mathbb{D}$ whether any other message $m\neq m^\star$ in such that $H_s(m)=H_s(m^\star)$. If we find one we output $\langle m,m^\star\rangle$, otherwise we output a nothing (a random generated value).

We have stored in the database all the collisions did previously.

From the way we’ve designed C_A it is easy to realise that

$$\Pr\big(\text{HasColl}_{C_A,\Pi}(n)=1\big)=\Pr(\text{coll}_A)$$

C_A is the pivot to solve the problem.

(ii) In the second reduction, we instead want to build an adversary B_A for Π using A as a subroutine. Our objective is, of course, to build B_A in such a way that

$$\Pr\Big(\mathsf{MacForge}_{B_A,\Pi}(n) = 1\Big) = \Pr\big(\mathsf{MacForge}_{A,\Pi^H}(n) = 1 \wedge \neg\mathrm{coll}_A\big) \qquad (\star)$$

we have to design B_A using A as a subroutine. B is supposed to be able to access to an oracle for $\mathsf{Mac}_k(\cdot)$ and it has a security param 1^n.

- First of all, generate a public seed by calling $\mathsf{Gen}^H(1^n)$.
- We can call the adversary A on 1^n.
- While running, A may call the oracle for $\mathsf{Mac}_k^H(\cdot)$. If the parameter of the query is m, we pass $H_s(m)$ to our own oracle, which of course returns a tag t which is forwarded to A.
- A outputs $\langle m^\star, t^\star\rangle$ and we cannot pass them as a result. m is a message which is not hashed yet. So we have to pass $m^\star$ to H, this way obtaining $H_s(m^\star)$. We can then return $\langle H_s(m^\star), t^\star\rangle$.

We have to prove that the equality $(\star)$ holds. We do that by showing that the two probabilistic events under considerations are actually the same probabilistic events.

$(\Rightarrow)$ Suppose that B_A wins against Π. This means that also A wins, because Mac^H is defined as $\mathsf{Mac}^H_{\langle s,k\rangle}(m) = \mathsf{Mac}_k(H_s(m))$. Moreover, if B_A wins, $H_s(m^\star)$ is different from any of the queries B_A made. So this also implies that $\neg\mathrm{coll}_A$ holds, because a collision in that sense of coll_A is precisely witnessed by $m \neq m^\star$ such that $H_s(m) = H_s(m^\star)$.

$(\Leftarrow)$ Suppose that A wins and further suppose that coll_A does not hold. This implies that $m^\star$ as produced by A is different from all the m on which A queries its oracle, and that $H_s(m^\star) = H_s(m)$ for any such m (because of $\neg\mathrm{coll}_A$), so we are in presence of a successful attack in the sense of B_A.

We then have two probabilistic events which hold in precisely the same situations, so $(\star)$ holds.

Without coll_A this proof would be broken. The not-collision status is crucial. It is essential to find a good pivot for a proof.

■

5. Constructing pseudorandom objects and hash functions in practice

5.1. Stream ciphers

Stream ciphers is usually built by a pair of algorithms:

- Init: initializes the internal state with some seed starting from a key (sometimes also with IV).
- GetBits: outputs a single bit modifying its state.

These two algorithms above are not defined to be random. How could we guarantee that?

5.1.1. Linear Feedback Shift Register

A state is n bits long $s_{n-1}s_{n-2}\cdots s_1s_0$. GetBits computes a bit b as the xor of some of the bits and returns in output the least significant bit shifting all the others to the right. Then s_{n-1} becomes b.

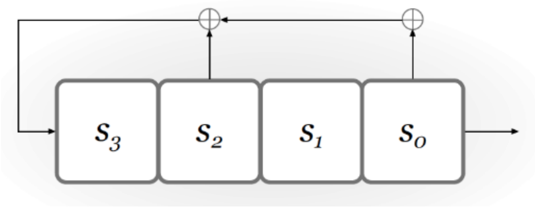


Figure 8: Linear shift

This is not random in some cases. Each bit of the seed is in the feedback loop. We can say that LFSR is not secure.

- The behaviour is determined by its state.
- With 2^n states, LFSR cycles through 2^{n-1} states.
- Statistically speaking it is quite good.
- From the first $2n$ outputs $y_1, ..., y_{2n}$ it is possible to totally reconstruct the initial state (i.e. $y_1, ..., y_n$) and the feedback coefficients (by means of a system of linear equations, which can be solved efficiently). Its linearity given by the feedback (just some xor) is too efficient to compute.

5.1.2. Trivium

It still have states but the feedback loop is a bit more complicated. It uses *AND* which prevents the circuit to be analysed linearly. It is used in practice but it could be broke.

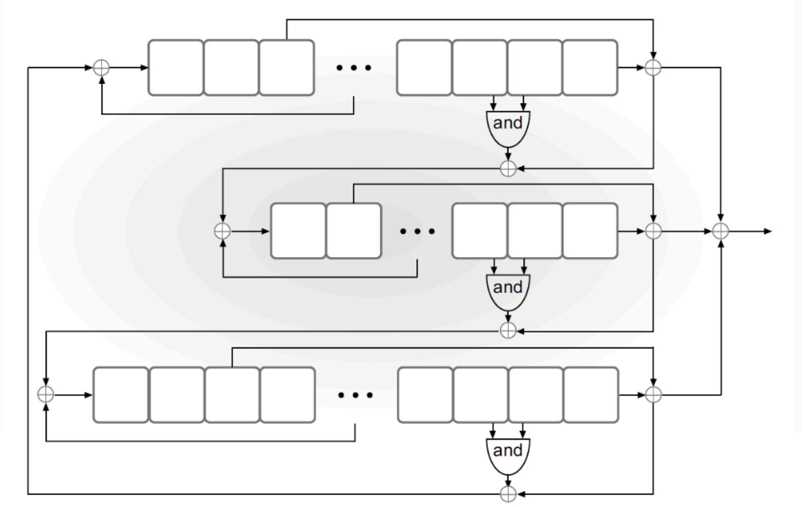


Figure 9: Trivium

5.1.3. RC4

It is used by Linux and it is implemented on software (not hardware). We have an internal state with a vector of bytes with a length of 256: this vector represents a permutation of the set $\{0, ..., 255\}$ with two other values $i, j \in \{0, ..., 255\}$.

Despite some statical attacks we can say that RC4 is not secure.

5.2. Block ciphers

Such a different story. We deal with function with the form $F : \{0,1\}^n \times \{0,1\}^\ell \mapsto \{0,1\}^\ell$ with n can be $\neq \ell$ and they're constants. Attacks against block ciphers are kind of ciphertext-only, known-ciphertext, chosen-plaintext, chosen-ciphertext. For the Kerchoff's principle, determining the key is sufficient to force the cipher.

5.2.1. Substitution-Permutation Network

A model with 3 phases:
- **The mixing with the key**. You can do a xor or taking a subset of.
- **S-BOX**. Chunk the key with smaller data (such as 6 bits) and apply a function for each chunk. Be careful when you write the truth table.
  - It must be invertible (it's the only way to go back from the 64-bit output, except for the xor if you fix one of the arg, of course).
  - If you change 1 bit of the input, at most 8 bits (1 chunk) would be different because all the other ones are unchanged (this is because you have many rounds).
- **Permutation**. Not compute the key, just permute.

The same transformation is applied more than once.

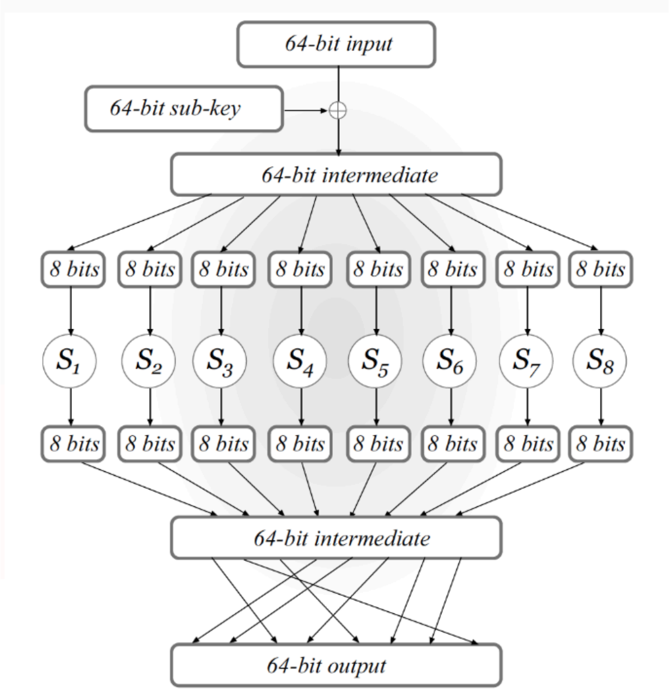


Figure 10: SPN

5.2.2. Feistel Network

It's adopted in practice. The message is cutted in two parts. A better way to understand what it does is to take a look at Figure 11. It is not invertible at all, it depends on $f_{1|2|3}$ that are not supposed to be invertible.

So, a message m is splitted in two chunks of sizes m_L and m_R.

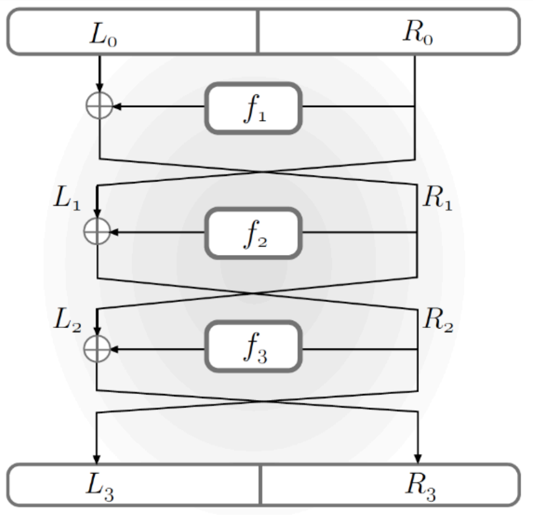


Figure 11: Feistel Network

A function f which depends on the key is called Mengler function.

5.2.3. DES

Its key of 56 bits is too low, so it has been broken. It's defined with

$$F_{\text{DES}} : \{0,1\}^{56} \times \{0,1\}^{64} \mapsto \{0,1\}^{64}$$

It is constructed as a 16-round Feistel network: for each $i \in \{1, ..., 16\}$ there exists $S_i \subseteq \{1, ..., 16\}$ of bits of the key that will contribute to the computation of f at round i.

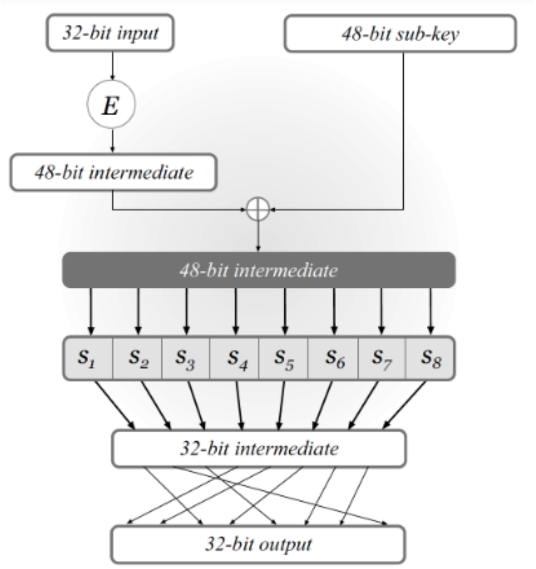


Figure 12: DES Mengler functions

In Figure 12 there are 8 S-BOX, each of them is a function $\{0, 1\}^6 \mapsto \{0, 1\}^4$ designed to be avalanche effect (the only ones responsible). Each possible configuration $s \in \{0, 1\}^4$ is the image of exactly four input configurations.

5.2.3.1. 3DES

This is not a triple encryption, but a key 112 bits long.

5.2.4. AES

It is a SPN once again with a message length of 128 bits and key of 128 bits (there are 3 versions, which use $128, 192$ or 256 bits for the key).

In each round, the state is seen as a 4×4 matrix of bytes, initially equal to the message $4 \cdot 4 \cdot 8 = 128$. There are 4 transformations for each:

1. **AddRoundKey**. A 128 subkey put in xor with the state.
2. **SubBytes**. One single S-BOX. Each byte in the matrix is replaced with the byte obtained by a fixed S-BOX 8×8.
3. **ShiftRows**. Each row of the matrix is shifted to the left of a variable number of positions. There is no permutation easy predicted.
4. **MixColumns**. No permutation but an invertible linear transformation for each column.

5.3. Constructing Hash Functions

They are very useful on authentication, called Hash MAC (or HMAC). Practically speaking they're not "keyed". In most cases, they're constructed from a compression function $C : \{0, 1\}^{\ell+n} \mapsto \{0, 1\}^{\ell}$. You obtain something similar to the Merkle-Damgard transformation.

Using a construction named **Davies-Meyer Construction**, given a block cipher $F : \{0, 1\}^n \times \{0, 1\}^{\ell} \mapsto \{0, 1\}^{\ell}$, a natural way to construct a compression function from F is to define $C : \{0, 1\}^{\ell+n} \mapsto \{0, 1\}^{\ell}$ as

$$C(k \parallel m) = F_k(m) \oplus m$$

Theorem If F is modelled as an ideal cipher, then the Davies-Meyer construction induces a collision-resistant hash-function in a concrete sense: each attacker of complexity q cannot find collisions with probability greater than $\frac{q^2}{2^\ell}$.

That is the same thing seen for the birthday attack. So, this is the probability of a brute attack against an hash function.

5.3.1. MD5

It is not considered cryptographically secure. Output is 128 bits long. Still used for checksums.

5.3.2. SHA

In SHA1 the output is 160 bits long, with a possible attack of 2^{80} function calls. Not really secure.

In SHA2 the output can be 256 or 512 bits long.

All of them use Davies-Meyer construction.

In SHA3 the output can be 256 or 512 bits long but the construction is different than the two SHA mentioned before.

6. Constructing pseudorandom objects and hash functions in theory

In this theoretical approach we show that pseudorandom objects can be constructed from the other objects whose existence is considered to be highly probable.

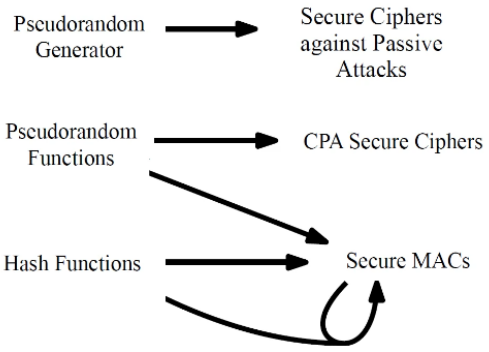


Figure 13: The situation in brief

The first example of Figure 13 is Π^G.

But we need to find something on the left for all of them.

6.1. One-Way functions

Easy to compute, hard to invert.

```
$\mathsf{Invert}_{A,f}(n)$ :
    $x \leftarrow \{0,1\}^n$
    $y \leftarrow f(x)$
    $z \leftarrow A(1^n, y)$
    return $f(z) \overset{?}{=} y$
```

You ask to the challenger to invert a function. z can be different than x.

Definition A function $f : \{0,1\}^\star \to \{0,1\}^\star$ is a one-way function $\iff$
1. $\exists$ a polytime and deterministic algorithm which computes f
2. $\forall$ PPT A, $\exists \varepsilon$ such that $\Pr\big(\mathsf{Invert}_{A,f}(n) = 1\big) \leq \varepsilon(n)$

One-way permutations are length-preserving one-way functions with the property that $y \in \{0,1\}^\star$ uniquely determines x such that $f(x) = y$.

Some functions assumed to be one-way:
- Multiplication between natural numbers.
- Subset-sum problem.

A one-way function does not entirely reveal x but this not implies the same to parts of x. Given a one-way function f, we can construct $g(x, y) = (x, f(y))$. g is also one-way function because if we could invert g, we can also invert f. But, $g(x, y)$ reveals a part of f (I mean x).

Definition (Hard-core predicate) A predicate $hc : \{0,1\}^\star \mapsto \{0,1\}$ is called hard-core predicate of a function f iff hc is polytime computable and $\forall$ PPT A holds

$$\Pr(A(f(x)) = hc(x)) \leq \frac{1}{2} + \varepsilon(n)$$

The adversary can't be able to find the value of x computing $hc(x)$.

For some non-one-way functions it is possible to construct trivial hard-core predicates. Eg. $f : \{0,1\}^\star \mapsto \{0,1\}^\star$ defined by $f(\varepsilon) = \varepsilon$ and $f(b \cdot s) = s \; \forall b \in \{0,1\}, s \in \{0,1\}^\star$, the result of $f(s)$ does not depend on the first bit of s, which can then become a hard-core predicate.

Theorem (Goldreich-Levin) If there's a one-way function (resp, a one-way permutation) f, then there exist a one-way function (resp, a one-way permutation) g and a hard-core predicate hc for g.

We do not prove this theorem because it is too difficult for this course.

The function g is constructed from f by setting $g(x,r) = (f(x), r)$ while hc is defined by $hc(x,r) = \oplus_{i=1}^{n} x_i \cdot r_i$. So, we're mixing some parts of first string with some parts of the second one.

Theorem (PRG from one-way permutation) f one-way permutations and hc hard-core predicate for f. Then, G defined by $G(s) = (f(s), hc(s))$ is a PRG with $\ell(n) = n+1$.

By definition, $hc(s)$, even if you know $f(s)$, is very hard to guess.

Theorem (Arbitrary Expansion Factor) If G is a PRG with $\ell(n) = n+1$ then exists a PRG H with an arbitrary expansion factor as long as it polynomial.

From s, you have $G(s)$ which stretchs the output a bit, but this "recursively" as seen in Figure 14.

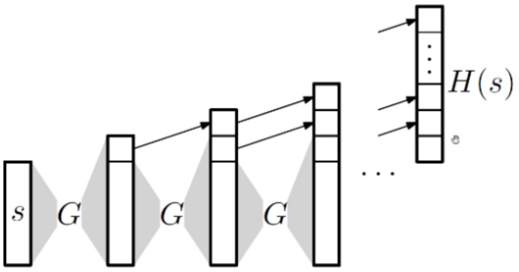


Figure 14: From PRG G to PRG H

Theorem (From generators to functions) PRG G with $\ell(n) = 2n$ then exists a PRF.

This is an issue of efficiency, and that's because it is not used. All of the branches of Figure 14 are independent.

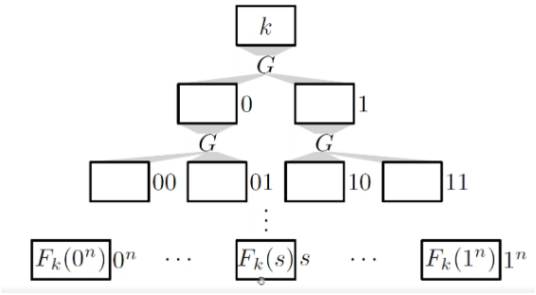


Figure 15: From PRG G to a PRF

Theorem If it exists a PRF then exists a strong PRF.

Theorem If it exists a PRG then exists a one-way function.

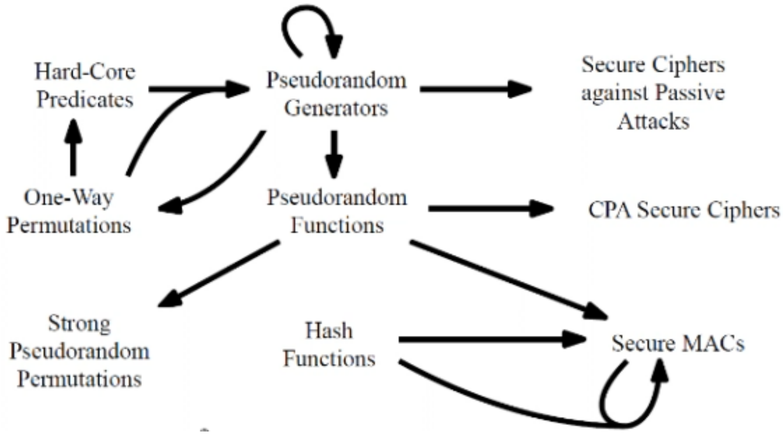


Figure 16: Overall picture

7. Algebra number theory and related assumptions

The following number theory refers to all those numbers in set $\mathbb{N}$ and $\mathbb{Z}$. We use abstract algebra, in particular the definition of groups.

The security of something depends on assumptions derived from number theory.

- $a, b \in \mathbb{Z}$, $a|b$ means $\exists c \in \mathbb{Z}$ such that $a \cdot c = b$.
- $a|b, a > 0$ means a is divisor of b.
- $p \in \mathbb{N}$ is prime if there do not exists factors of it. Otherwise is named composite.
- $n, m \in \mathbb{N}, m > 1$, $n \bmod m$ is the reminder of $\frac{n}{m}$.

Lemma If $n, m \in \mathbb{N}, m > 1 \Rightarrow n$ is invertible $\bmod m$, i.e. $\exists p$ such that $n \cdot p \bmod m = 1$ whenever $\gcd(n, m) = 1$. i.e m, n are coprime.

Factoring (find a $N = p \cdot q$) is an hard problem, something like brute force, because we want to invert a multiplication. If we use a classical algorithm which has $O\left(\sqrt{N}(\log N)^k\right)$ [k is a constant very small, like 3 that depends on the logarithm; we stop to something less than $\sqrt{N}$] we have something polynomial on N but not on the length of N. We want to care about the length of the number.

```
wFactor_A(n) :
    $(x, y) \leftarrow \mathbb{N} \times \mathbb{N}$ with $|x| = |y| = n$
    $N \leftarrow x \cdot y$
    $(z, w) \leftarrow A(N)$
    return $z \cdot w \overset{?}{=} N$
```

An adversary is responsible to find $z(= x)$ and $w(= y)$. The adversary A succeeds if $\exists \varepsilon \in \mathcal{NGL}$ such that $\Pr(\mathsf{wFactor}_A(n) = 1) = \varepsilon(n)$.

If either x or y is even, then N will be even. This does not align to the definition of hard problem. N will be even with a probability of $\frac{3}{4}$.

But, the product of two prime numbers assures to have only one way to factorize.

```
GeneratePrimeNumbers(n):        # n is the num bits
    for $i \leftarrow 1$ to $t$ do
        $r \leftarrow \{0, 1\}^{n-1}$
        $p \leftarrow 1 \parallel r$
        if $p$ is prime then
            return $p$
    panic()
```

All numbers with n bits have the same probability to be generated.

The last bit of p is always 1. We can be sure to have a string of length n.

Theorem (Density of a number) There exist a constant c such that for every $n > 1$ the number of primes that can be represented in exactly n bits is at least equals to $\frac{c \cdot 2^{n-1}}{n}$.

The number above is exponential. When we fix the number of bits, we find that there is at least one prime number. More specifically, there is an exponential quantity of them.

Given that theorem you can be able to value t.

So, the probability that p is prime will be at least equal to $\frac{\text{Number of successful cases}}{\text{Number of all cases}}$:

$$\frac{\frac{c \cdot 2^{n-1}}{n}}{2^{n-1}} = \frac{c \cdot 2^{n-1}}{n \cdot 2^{n-1}} = \frac{c}{n}$$

The fraction $\frac{c}{n}$ tends to 0 as $n \to \infty$, but it does very slowly. Therefore, we need to account for the fact that this fraction is not negligible.

Thus, if $t = \frac{n^2}{c}$ the probability to fail is, whenever $n \geq c$, at most equal to

$$\left(1 - \frac{c}{n}\right)^t = \left(\left(1 - \frac{c}{n}\right)^{\frac{n}{c}}\right)^n \leq \left(e^{-1}\right)^n = e^{-n}$$

where $\left(1 - \frac{c}{n}\right)^t$ represent the bound of the probability of failure.

There are deterministic algorithms such as AKS to test the primality of a number in polynomial time but also some probabilistic such as the Miller-Rabin test. The latter gets a prime number in input and returns OK with probability 1 but if the input is a composite number returns NO with probability $1 - \varepsilon(|p|)$.

The old experiment wFactor can use an algorithm GenModulus which, given a string 1^n on input, returns a triple (N, p, q) where $N = p \cdot q$ and p, q are primes with n bits. From this, we can build

```
GenModulus($1^n$):
    $p \leftarrow \{0,1\}^n$        # $p$ is prime
    $q \leftarrow \{0,1\}^n$        # $q$ is prime
    $N \leftarrow p \cdot q$
    return $(N, p, q)$
```

```
Factor$_{A,\ \text{GenModulus}}(n)$:
    $(N, p, q) \leftarrow$ GenModulus($1^n$)
    $(r, s) \leftarrow A(N)$
    return $r \cdot s \overset{?}{=} N$
```

Thanks to GenModulus we can say that factoring is hard relative $\iff \forall A$ PPT $\exists \varepsilon \in \mathcal{NGL}$ such that

$$\Pr\bigl(\mathsf{Factor}_{A,\ \mathsf{GenModulus}}(n) = 1\bigr) = \varepsilon(n)$$

This assumption is sufficient to obtain a one-way function but not to prove the security of public-key schemes, because quantum computer can be easy able to obtain what we want.

7.1. Group Theory

A group is an algebraic structure $(\mathbb{G}, \circ)$ where $\circ$ is a binary operation that is associative, with identity e and where every $g \in \mathbb{G}$ has an inverse g^{-1}.

A finite group is said to have order equal to $|\mathbb{G}|$.

The $\circ$ operation is named bullet. If commutative is said to be abelian.

Integer and natural numbers are not considered for this.

We can see the number as

- additive +: $mg = \underbrace{g + \ldots + g}_{m \text{ times}}$
- multiplicative $\cdot$: $g^m = \underbrace{g \cdot \ldots \cdot g}_{m \text{ times}}$

7.2. Exponentiation

The computation of $m \cdot g$ or g^m can be performed in a number of operations which is polynomial in $|m|$, i.e logarithmic in m. A fast exponentiation used in cryptography is the binary representation of m.

For example, $g^{11} = g^8 \cdot g^2 \cdot g^1 = g^{2^3} \cdot g^{2^1} \cdot g^{2^0}$ and each of the factors, which are at most $|m|$ can be computed in time that is linear in $|m|$.

Theorem If $(\mathbb{G}, \circ)$ is a finite group where $|G| = m$ is the order of the group then for each $g \in \mathbb{G}$, it is true that $g^m = 1_{\mathbb{G}}$.

Proof

Let's prove the theorem in the special case in which the group is abelian.

Suppose that $\mathbb{G} = \{g_1, g_2, \ldots, g_m\}$ and $g_i \neq g_j$ when $i \neq j$. Let's fix any element $g \in \mathbb{G}$. We want to prove

$$g_1 \cdot g_2 \cdot \ldots \cdot g_m = (gg_1) \cdot (gg_2) \cdot \ldots \cdot (gg_m) \qquad (\star)$$

By contradiction, all the elements on the right side are distinct because $gg_i = gg_j \Rightarrow g^{-1}(gg_i) = g^{-1}(gg_j)$

$$\begin{aligned} &\Rightarrow (g^{-1}g)g_i = (g^{-1}g)g_j \\ &\Rightarrow g_i = g_j \\ &\Rightarrow i = j \end{aligned}$$

Since $(\mathbb{G}, \circ)$ is abelian, we can rearrange $(\star)$ as

$g_1 \cdot g_2 \cdot \ldots \cdot g_m = g^m \cdot (g_1 \cdot g_2 \cdot \ldots \cdot g_m)$

we can multiply both sides by $(g_1 \cdot g_2 \cdot \ldots \cdot g_m)^{-1}$ obtaining

$(g_1 \cdot \ldots \cdot g_m) \cdot (g_1 \cdot \ldots \cdot g_m)^{-1} = g^m \cdot (g_1 \cdot \ldots \cdot g_m) \cdot (g_1 \cdot \ldots \cdot g_m)^{-1} \Rightarrow 1_{\mathbb{G}} = g^m \cdot 1_{\mathbb{G}} \Rightarrow g^m = 1_{\mathbb{G}}$

■

You can't multiply some increased numbers and get back a 1.

Theorem If $(\mathbb{G}, \circ)$ has order $m > 1$, then for every $g \in \mathbb{G}$ and for every i, $g^i = g^{[i \bmod m]}$.

So you compute g power the remainder. It's quite easy to work in the group.

7.3. Euler function

The Euler function is $\Phi(N) = |\mathbb{Z}_N^\star|$, which is the number of how many elements are prime. Thus, $1 \leq \Phi(N) < N$.

- N prime number $p \Rightarrow$ every $1 \leq n < p$ is coprime with p.
  - $\Phi(N) = p - 1$.
- N product of two primes $p, q \Rightarrow \gcd(a, N) \neq 1$ $(p|a \vee q|a)$.
  - $$\begin{aligned}\Phi(N) = N - 1 - (p-1) - (q-1) = \\ p \cdot q - p - q + 1 = \\ p(q-1) - 1(q-1) = \\ (p-1) \cdot (q-1)\end{aligned}$$

7.4. Finite groups example

1. $\mathbb{Z}_N = \{0, ..., N-1\}$ is a group if the underlying operation is addition modulo N, i.e. the map matching (a, b) with $a + b \bmod N$. The identity is 0 and the inverse of $n \in \mathbb{Z}_N$ is $N - m$. But we can use this set with multiplication modulo N when we eliminate 0 because it is not invertible and N is prime: the latter guarantees that every $1 < n < N$ is invertible modulo N.

   For $\mathbb{Z}_N$ with composite N there's a way to make the set a group with respect to the multiplication modulo N, using a $\mathbb{Z}_N^\star \subseteq \mathbb{Z}_N$, where $\mathbb{Z}_N^\star = \{n \in \mathbb{Z}_N \mid \gcd(n, N) = 1\}$.

   Both groups are abelian.

2. $\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$ $\qquad (\mathbb{Z}_5, \overline{+})$ where $\overline{+}$ is addition modulo 5.

   e.g. $3 \,\overline{+}\, 4 = 3 + 4 \bmod 5 = 2 \qquad 4 \,\overline{+}\, 1 = 4 + 1 \bmod 5 = 0$.

3. $\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$ $\qquad (\mathbb{Z}_5, \bar{\cdot})$

   e.g. $3 \,\bar{\cdot}\, 2 = 3 \cdot 2 \bmod 5 = 1 \qquad 4 \,\bar{\cdot}\, 4 = 16 \bmod 5 = 1 \qquad 1 \,\bar{\cdot}\, 3 = 3 \bmod 5 = 3$.

   But, 0 shouldn't be considered in this group for this operation. So, if we use $\mathbb{Z}_5^\star = \mathbb{Z}_5 \setminus \{0\}$ we have $\Phi(5) = 4$.

4. $\mathbb{Z}_6 = \{0, 1, 2, 3, 4, 5, 6\}$ $\qquad (\mathbb{Z}_6, \bar{\cdot})$

   e.g. $5 \,\bar{\cdot}\, 5 = 25 \bmod 6 = 1 \qquad 1 \,\bar{\cdot}\, 1 = 1$ can't use 0, so $\mathbb{Z}_6$ is not a group.

   $\mathbb{Z}_6^\star = \{1, 5\}$. So $\Phi(6) = 2$.

7.5. Cardinality of $\mathbb{Z}_N^\star$

The Euler function is defined as $\Phi(N) = |\mathbb{Z}_N^\star|$. Also, $1 \leq \Phi(N) < N$.

Theorem Let $N > 1$. For every $e \in \mathbb{N}, e > 0$, we define $f_e : \mathbb{Z}_N^\star \mapsto \mathbb{Z}_N^\star$ assuming $f_e(x) = x^e \bmod N$. If $\gcd(e, \Phi(N)) = 1$, then f_e is a permutation. Moreover, if d is the inverse of e (modulo $\Phi(N)$), then f_d is the inverse of f_e.

e is coprime with $\Phi(N)$.

f_e is a bijective function, because you can invert it.

f_e is efficiently computable from x if you know both e and N.

Given e and $\Phi(N)$ the inverse d of e modulo $\Phi(N)$ is efficiently computable.

Given N as product of two primes p, q, the value $\Phi(N)$ is not easily computable.

7.6. RSA assumption

The factorization issue becomes easy to compute if we know which prime numbers p, q form N or if we know the inverse number d of the number e modulo $\Phi(N)$, in other words $\gcd(e, \Phi(n)) = 1$.

As assumption we have that GenRSA returns:

- $N = p \cdot q$ with $|p| = |q| = n$.
- $e \in \mathbb{N}$ s.t. $\gcd(e, \Phi(N)) = 1$.
- $d \in \mathbb{N}$ s.t. $e \cdot d \bmod \Phi(N) = 1$.

We can create an experiment which uses an attacker A.

```
$\mathsf{RSAInv}_{A,\ \mathsf{GenRSA}}(n)$:
    $(N, e, d) \leftarrow \mathsf{GenRSA}(1^n)$
    $y \leftarrow \mathbb{Z}_N^\star$
    $x \leftarrow A(N, e, y)$
    return $x^e \bmod N \stackrel{?}{=} y$
```

```
$\mathsf{GenRSA}(1^n)$:
    $(N, p, q) \leftarrow \mathsf{GenModulus}\ (1^n)$
    $M \leftarrow (p-1)(q-1)$
    $e \leftarrow \{1, ..., M\}$        # such that $\gcd(e, M) = 1$
    $d \leftarrow e^{-1} \bmod M$
    return $(N, e, d)$
```

$$\text{RSA is hard relatively to } \mathsf{GenRSA}$$

$$\Updownarrow$$

$$\forall A \text{ PPT}, \exists \varepsilon \in \mathcal{NGL}.\, \Pr\big(\mathsf{RSAInv}_{A,\ \mathsf{GenRSA}}(n) = 1\big) = \varepsilon(n)$$

RSA assumption is stronger than factoring assumptions.

7.7. Cyclic groups

Given a finite multiplicative group $(G, \cdot)$, one of its elements $g \in \mathbb{G}$ and construct

$$\langle g \rangle = \{g^0, g^1, ...\} \subseteq \mathbb{G}$$

This group is said to be cyclic if $\exists g \in \mathbb{G}$ with $\langle g \rangle = \mathbb{G}$. Also, g is said generator of $\mathbb{G}$.

As previously proved, $g^m = 1_\mathbb{G}$ so we can write $\langle g \rangle = \{g^1, ..., g^m\}$. This infinite sequence is periodic; the sequence can't be infinite because the finite hypothesis of $\mathbb{G}$.

There could be a $i < m$ such that $g^i = 1_\mathbb{G}$. With $\langle g \rangle = \{g^1, ..., g^i\}$ there would be i elements: if $1 \leq k < j < i$ then $g^j = g^k \Rightarrow \frac{g^j}{g^k} = 1_\mathbb{G} \Rightarrow g^{j-k} = 1_\mathbb{G}$

The cardinality of $\langle g \rangle$, or order of $g \in \mathbb{G}$, is the smallest $i \in \mathbb{N}$ such that $g^i = 1$.

Lemma If $\mathbb{G}$ has order m and $g \in \mathbb{G}$ has order i, then $i | m$.

Proof

If this were not be possible, we'd have $m = ik + j$ with $j < i$. But then

$$g^j = g^{j+ik-ik} = g^{m-ik} = g^m \left(g^i\right)^{-k} = 1_{\mathbb{G}} \left(1_{\mathbb{G}}\right)^{-k} = 1_{\mathbb{G}}$$

■

Theorem If $\mathbb{G}$ has prime order then $\mathbb{G}$ is cyclic and every $g \in \mathbb{G}$ with $g \neq 1_{\mathbb{G}}$ generates $\mathbb{G}$.

7.8. Discrete logarithm

If $\mathbb{G}$ is a cyclic multiplicative group, then there exist a "natural" biunivocal correspondence between $\mathbb{G}$ and $\mathbb{Z}_{|\mathbb{G}|}$.

Every $h \in \mathbb{G}$ can be matched with a unique $x \in \mathbb{Z}_{|\mathbb{G}|}$ that is the one such that $g^x = h$. Now, x is the discrete log of h with respect to g, which we write $\log_g h$.

The computing of $\log_g h$ given a cyclic group G which uses a generator g for $\mathbb{G}$ and a random element h is the discrete logarithm problem.

The routine GenCG with input 1^n constructs a group $\mathbb{G}$, of order q, with $|q| = n$, and a generator $g \in \mathbb{G}$.

```
$\mathrm{DLog}_{A,\ \mathsf{GenRSA}}(n)$:
    $(\mathbb{G}, q, g) \leftarrow \mathsf{GenCG}(1^n)$
    $h \leftarrow \mathbb{G}$        # random value from the group
    $x \leftarrow A(\mathbb{G}, q, g, h)$
    return $g^x \overset{?}{=} R$
```

This is valid iff $\forall A$ PPT, $\exists \varepsilon \in \mathcal{NGL}. \Pr\big(\mathrm{DLog}_{A,\ \mathsf{GenRSA}}(n) = 1\big) = \varepsilon(n)$.

7.9. Computational Diffie-Hellman (CDH) assumption

Given a cyclic group $\mathbb{G}$ and a generator $g \in \mathbb{G}$, we define a function $DH_g : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}$ as $DH_g(h, j) = g^{(\log_g h) \cdot (\log_g j)}$.

1. $DH_g(g^x, g^y) = g^{xy} = (g^x)^y = (g^y)^x$
2. CDH wants to compute efficiently DH_g.
3. CDH assumption holds if CDH is hard.
4. Each efficient algorithm for the discrete logarithm problem generates an efficient algorithm for CDH.

7.10. Decisional Diffie-Hellman (DDH) asumption

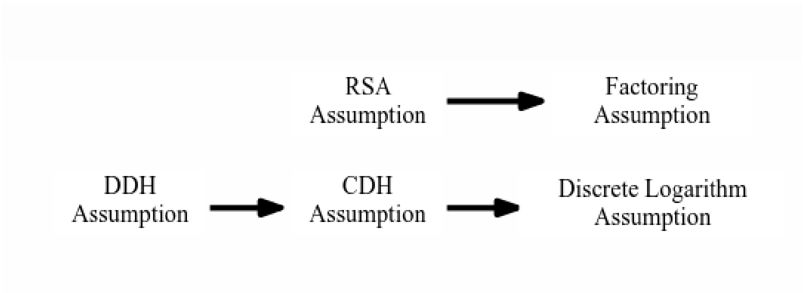
The problem DDH consists in distinguishing $DH_g(h, j)$ from an arbitrary element of the group $\mathbb{G}$, given obviously h and j.

Formally, DDH is hard iff $\forall A$ PPT, $\exists \varepsilon \in \mathcal{NGL}$ s.t.

$$|\Pr(A(\underbrace{\mathbb{G}}_{\text{group}}, \underbrace{q}_{\text{order}}, \underbrace{g}_{\text{generator}}, g^x, g^y, g^z) = 1) - \Pr(A(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1)| \leq \varepsilon(n)$$

g^x, g^y and g^z are random values from the group.

$(\mathbb{G}, q, g)$ is produced by $\mathsf{GenCG}(1^n)$.



7.11. Diffie-Hellman assumptions on specific groups

It should be noted that the use of groups with a prime number of elements is to be preferred, this is because:

1. It's trivial to test if an element is or is not a generator.
2. If $\mathbb{G}$ is a group of prime order q with $|q| = \Theta(2^n)$ then $\Pr\big(DH_g(h,j) = y\big) = \frac{1}{q} + \varepsilon(n)$.

If we consider the groups $\mathbb{Z}_p^\star$ where p is a prime.

1. An algorithm GenCG that generates groups of this type exists and it is efficient. The discrete logarithm assumption applies to these groups.
2. DDH is not believed to be hard for these groups.
3. However, there is a different algorithm GenCG which returns a subset of $\mathbb{Z}_p^\star$ and for which DDH is also believed to be hard.

7.12. From factoring to 1-way functions

Consider a function GenModulus that takes as input at most $p(n)$ random bits of length n, where p is a polynomial.

We construct an algorithm that computes a function $f_{\mathsf{GenModulus}}$ as follows:

1. The input is a string x;
2. Compute an integer n such that $p(n) \leq |x| \leq p(n+1)$;
3. Compute (N, p, q) as the result of $\mathsf{GenModulus}(1^n)$ using as random bits those in x, that are enough.
4. Return N.

We observe now how the following distributions are identical for each $m \in N$.

1. The result N of $f_{\mathsf{GenModulus}(x)}$ where $x \in \{0,1\}^m$ is randomly chosen.
2. The result N of $\mathsf{GenModulus}(1^n)$ where $p(n) \leq m \leq p(n+1)$.

Theorem If factoring is hard relative to GenModulus, then $f_{\mathsf{GenModulus}}$ is a one-way function.

8. Public-Key Encryption

We have two keys:

- encryption key, used by the sender.
- decryption key, user by the receiver.

Key distribution can be on public channels, each of the parties involved must keep secret a single key, each new user creates a new public key.

8.1. Key Exchange

Set of rules Π that specifies how two parties A and B should exchange messages.

The security is formulated through an experiment, and this captures only passive adversaries.

The transcript of a protocol is the content of all the messages exchanged by the parties. It also contains the key k.

```
$\mathsf{KE}^{eav}_{A,\Pi}(n)$:
    $(\text{transcript}, k) \leftarrow \Pi(1^n)$
    $b \leftarrow \{0,1\}$
    **if** $b = 0$ **then**
        $k^\star \leftarrow \{0,1\}^n$
    **else**
        $k^\star \leftarrow k$
    $b^\star \leftarrow A(\text{transcript}, k^\star)$
    **return** $\neg(b \oplus b^\star)$
```

Π is secure iff for every A PPT $\exists \varepsilon \in \mathcal{NGL}$ s.t. $\Pr\left(\mathsf{KE}^{eav}_{A,\Pi}(n) = 1\right) = \frac{1}{2} + \varepsilon(n)$.

8.1.1. Diffie-Hellman protocol

It exploits the things of cyclic groups. Given the two parties we have:

A:

```
$(\mathbb{G}, q, g) \leftarrow \mathsf{GenCG}(1^n)$
$x \leftarrow \mathbb{Z}_q$
$h_1 \leftarrow g^x$
**send** $(\mathbb{G}, q, g, h_1)$ **to** $B$
```

B (which does not know x):

```
$y \leftarrow \mathbb{Z}_q$
$h_2 \leftarrow g^y$
**send** $h_2$ to $A$
**return** $h_1^y$
```

A:

```
**return** $h_2^x$
```

It's like a partial kind of secrecy.

The exchange protocol like above is correct if $k_B = k_A$. Here we have

$$k_B = h_1^y = (g^x)^y = g^{xy}$$

$$k_A = h_2^x = (g^y)^x = g^{xy}$$

Theorem If DDH holds with respect to GenCG then the DH key-exchange protocol is secure against KE^{eav}.

It is related to discrete algorithm. The issue here is that KE^{eav} does not avoid situations like impersonification attacks or MitM attacks.

Proof

Let's just start from the expression we want to show bounded by $\frac{1}{2} + \varepsilon(n)$:

$\Pr\left(\mathsf{KE}^{eav}_{A,\Pi}(n) = 1\right) =$

$$\begin{aligned}
&= \Pr\left(\mathsf{KE}^{eav}_{A,\Pi}(n) = 1 \mid b = 0\right) \cdot \Pr(b = 0) + \Pr\left(\mathsf{KE}^{eav}_{A,\Pi}(n) = 1 \mid b = 1\right) \cdot \Pr(b = 1) \\
&= \tfrac{1}{2}\left[\Pr\left(\mathsf{KE}^{eav}_{A,\Pi}(n) = 1 \mid b = 0\right) + \Pr\left(\mathsf{KE}^{eav}_{A,\Pi}(n) = 1 \mid b = 1\right)\right] \\
&= \tfrac{1}{2}[\Pr(A(\text{transcript}, r) = 0) + \Pr(A(\text{transcript}, k) = 1)] \\
&= \tfrac{1}{2}[\Pr(A(\mathbb{G}, q, g, g^x, g^y, g^z) = 0) + \Pr(A(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1)] \\
&= \tfrac{1}{2}[(1 - \Pr(A(\mathbb{G}, q, g, g^x, g^y, g^z) = 1)) + \Pr(A(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1)] \\
&= \tfrac{1}{2} + \tfrac{1}{2}[\Pr(A(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1) - \Pr(A(\mathbb{G}, q, g, g^x, g^y, g^z) = 1)] \\
&= \tfrac{1}{2} + \tfrac{1}{2}\varepsilon(n)
\end{aligned}$$

The right side of the expression is negligible thanks to the DDH assumption. i.e. A function ε which is a negligible function.

The adversary for the protocol and the adversary for DDH are the same.

■

8.2. Asymmetrical scheme

We use a pair of keys $\langle pk, sk\rangle$ where $pk \neq sk$. The first of them is named "public key", the latter "secret key".

Here, there's not the problem about key sharing on private channel. Each user must manage the secrecy of only one key.

But, how can you authenticate a public key? We have to be sure about the ownership of a key.

For a scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ we have some edits:

- Gen takes in input a string 1^n and outputs a pair $\langle pk, sk\rangle$ such that $|pk|, |sk| \geq n$ and n can be inferred by one of them.
- Enc takes in input a message m and a public key pk.
- Dec can be probabilistic and takes in input a ciphertext c and a secret key sk.

For a correct scheme in a probabilistic sense, we have that there exist a negligible function ε such that for every pair $\langle pk, sk\rangle$ generated by a $\mathsf{Gen}(1^n)$ and a n,

$$\Pr(\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) \neq m)\mathsf{Enc} \leq \varepsilon(n)$$

The adversary knows the public key, and this is a different way to see the PubK.

```
$\mathsf{PubK}^{eav}_{A,\Pi}(n)$:
    $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$
    $(m_0, m_1) \leftarrow A(1^n, pk)$
    if $|m_0| \neq |m_1|$ then
        return 0
    $b \leftarrow \{0, 1\}$
    $c \leftarrow \mathsf{Enc}(pk, m_b)$
    $b^\star \leftarrow A(c)$
    return $\neg(b \oplus b^\star)$
```

This above is about a passive attacker (eavesdropper). Security about passive attacks and CPA are seen in the same way because an attacker A is able to encrypt any message because they can use the public key of the victim. Since a passive adversary has got access to pk, it is also considered active.

Theorem If Π is secure against passive attacks $\Rightarrow$ Π is secure against CPA.

The theorem of Shannon about security is not used because it can be used only for symmetrical key scheme.

Theorem There are no asymmetric ciphers that are secure in a perfect sense.

Theorem No public key scheme in which Enc is deterministic can be secure with respect to PubK^{eav}.

8.2.1. Multiple encryptions

Let's define a new experiment PubK^{mult} in which the adversary outputs not a pair of messages (m_0, m_1) but a pair of tuple of messages (m_0, m_1) where $m_0 = (m_0^1, ..., m_0^t), m_1 = (m_1^1, ..., m_1^t)$, and $|m_0^j| = |m_1^j|$.

Π is said to be secure with respect to multiple encodings iff for every PPT A there exist ε with

$$\Pr\left(\mathsf{PubK}^{mult}_{A,\Pi}(n) = 1\right) = \frac{1}{2} + \varepsilon(n)$$

Theorem If Π is secure with respect to PubK^{eav}, then it is secure with respect to PubK^{mult} .

8.2.2. Hybrid encryption

A public key scheme is less efficient than private key scheme, so we could combine them.

- $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is public key scheme.
- $\Pi' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ is private key scheme.
- Π^{H_y} is scheme in which the encryption is defined in Figure 18.

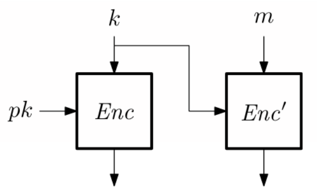


Figure 18: Hybrid encryption

This is what it's used in practice, such as into operating systems.

The key is what we expect from the public key.

```
$\mathsf{Gen}^{H_y}(1^n)$:
    return $\mathsf{Gen}(1^n)$
```

```
$\mathsf{Enc}^{H_y}(pk, m)$:
    $k \leftarrow \{0,1\}^n$
    $c \leftarrow \mathsf{Enc}_{pk}(k)$
    $d \leftarrow \mathsf{Enc}'_k(m)$
    return $(c, d)$
```

```
$\mathsf{Dec}^{H_y}(sk, (c, d))$:
    $k \leftarrow \mathsf{Dec}_{sk}((c, d))$
    $m \leftarrow \mathsf{Dec}'_k(d)$
    return $m$
```

Theorem If Π is CPA-secure and Π' has indistinguishable encryptions, then Π^{H_y} is secure.

The time you take to encrypt t bits is $\text{TIME}(t) = \frac{\alpha+\beta t}{t}$ where α is fixed for the key and it can be very large, β is the time to encrypt each bit.

$$\lim_{t\to\infty} \frac{\alpha + \beta t}{t} = \beta$$

As $|m|$ increases, the quantity $|c|$ stays constant, while there are private-key encryption schemes such that $|d| = |m| + n$. Therefore, as $|m|$ increases, the length of (c, d) is linear.

8.2.3. Textbook RSA

We define a scheme called Textbook RSA.

```
$\mathsf{Gen}(1^n)$:
    $(N, e, d) \leftarrow \mathsf{GenRSA}(1^n)$
    return $((N, e), (N, d))$
```

```
$\mathsf{Enc}((N, e), m)$:
    $c \leftarrow m^e \bmod N$
    return c
```

```
$\mathsf{Dec}((N, d), c)$:
    $m \leftarrow c^d \bmod N$
    return $m$
```

This is insecure, we do not have an Enc deterministic. There are a lot of attacks, just look at Cyberchallenge notes.

8.2.3.1. Secure version of RSA (Padded RSA)

A different scheme called Padded RSA is secure.

```
$\mathsf{Gen}(1^n)$:
    $(N, e, d) \leftarrow \mathsf{GenRSA}(1^n)$
    return $((N, e), (N, d))$
```

```
$\mathsf{Enc}((N, e), m)$:
    $r \leftarrow \{0, 1\}^{|N| - \ell(n) - 1}$
    $c \leftarrow (r \| m)^e \bmod N$
    return c
```

```
$\mathsf{Dec}((N, d), c)$:
    $m \leftarrow \mathrm{LSB}\big(c^d \bmod N, \ell(n)\big)$
    return $m$
```

$\ell(n)$ is a function such that $|m| \leq \ell(n) \leq 2n - 2$. It should be sufficiently small, less than linear.

We do have something randomized in encryption with value r.

Theorem If the RSA Assumption holds with respect to GenRSA and if $\ell(n) = O(\log n)$, then Padded RSA is secure with respect to passive attacks.

8.2.3.2. Elgamal

It is an encryption scheme based from the DDH assumption for passive attacks.

Fixed two $m, c \in \mathbb{G}$ of a finite group, the probability that a random element $k \in \mathbb{G}$ is such that $m \cdot k = c$ is equal to $\frac{1}{|\mathbb{G}|}$.

$$\Pr(m \cdot k = c) = \Pr(k = m - 1 \cdot c) = \frac{1}{|\mathbb{G}|}$$

```
$\mathsf{Gen}(1^n)$:
    $(\mathbb{G}, q, g) \leftarrow \mathsf{GenCG}(1^n)$
    $x \leftarrow \mathbb{Z}_q$
    $sk \leftarrow (\mathbb{G}, q, g, x)$
    $pk \leftarrow (\mathbb{G}, q, g, g^x)$
    return $(sk, pk)$
```

```
$\mathsf{Enc}((\mathbb{G}, q, g, h), m)$:
    $y \leftarrow \mathbb{Z}_q$
    return $(g^y, \underbrace{h^y \cdot m}_{\substack{h=g^x \\ h^y=(g^x)^y \\ h^y \cdot m=d}})$
```

```
$\mathsf{Dec}((\mathbb{G}, q, g, x), (c, d))$:
    return $\frac{d}{c_1^x}$
```

We're using the assumption that discrete algorithm calculation is hard.

The correctness of scheme is proved by

$$\frac{d}{c_1^x} = \frac{h^y \cdot m}{g^{yx}} = \frac{(g^x)^y \cdot m}{g^{xy}} = m$$

- It is often used in practice, such as cryptocurrencies, also for Zero-Knowledge Proofs.
- In Fully-Homeomorphic Encryption Schemes you delegate the storage to the cloud which is responsible to encrypt and decrypt files.
- In Post-Quantum Cryptography we use quantum computers.

Theorem If Assumption DDH holds with respect to GenCG, then the Elgamal scheme is secure.

Proof

Let us consider, just for the sake of proving this result, a variation $\tilde{\Pi}$ of the Elgamal encryption scheme, in which Gen is kept like in Elgamal, while Enc is replaced by the algorithm

```
$\widetilde{\mathsf{Enc}}((\mathbb{G}, q, g, h, m), m)$:
    $y \leftarrow \mathbb{Z}_q$
    $z \leftarrow \mathbb{Z}_q$
    return $(g^y, g^z \cdot m)$
```

Although being completely useless in practice, $\tilde{\Pi}$ satisfies the following property:

$$\Pr\left(\mathsf{PubK}^{eav}_{A,\tilde{\Pi}}(n) = 1\right) = \frac{1}{2}$$

Because the challenge ciphertext (the element c) contains no information allowing the adversary to discriminate between m_0 and m_1 simply because m is multiplied by y^z and the first component g^y is independent from g^z.

- Now the real proof by reduction can start.

We build an adversary B against DDH from an adversary A against Elgamal in such a way that if A is successful, then B is successful:

```
$B(\mathbb{G}, q, g, g^z, g^y, h)$:
    $(m_0, m_1) \leftarrow A\left(1^{|q|}, (\mathbb{G}, q, g, g^x)\right)$
    $b \leftarrow \{0, 1\}$
    $c \leftarrow (g^y, m_b \cdot h)$
    $b^\star \leftarrow A(c)$
    return $\neg(b \oplus b^\star)$
```

We know that

$$\Pr\left(\mathsf{PubK}^{eav}_{A,\tilde{\Pi}}(n) = 1\right) = \Pr(B(\mathbb{G}, q, g, g^x, g^y, g^z) = 1)$$

$$\Pr(\mathsf{PubK}^{eav}_{A,\underbrace{\Pi}_{\text{Elgamal}}}(n) = 1) = \Pr(B(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1)$$

If the h of B becomes g^z, it will be used on the calculation of c at line 3.

The first one probability is $\frac{1}{2}$ but the latter is far away to that value.

If A breaks Π, then $\Pr\left(\mathsf{PubK}^{eav}_{A,\Pi}(n) = 1\right)$ is in the form $\frac{1}{2} + \eta(n)$ where η is not negligible. But, since we know that $\Pr\left(\mathsf{PubK}^{eav}_{A,\tilde{\Pi}}(n) = 1\right) = \frac{1}{2}$, then we can conclude that

$$|\Pr(B(\mathbb{G}, q, g, g^x, g^y, g^z) = 1) - \Pr(B(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1)| = \eta(n)$$

which is the thesis. B is successful.

■

9. The Symbolic Model

It's an higher level of abstraction than what we've seen about binary strings. Also named Dolev-Yao.

In the computational model we used cryptographic primitives and protocols that are modelled using PPT algorithms, using a probability of success for an A of a negligible function ε. The computational model has got some limits: first of all, probabilistic reasonings becomes difficult for complex frameworks; also, for an increased number of participants it decreases the efficiency. Examples of issues like those are cryptocurrencies and e-commerce.

9.1. Needham-Schroeder Protocol

It uses A, B which are the participant IDs and nonces N_A, N_B

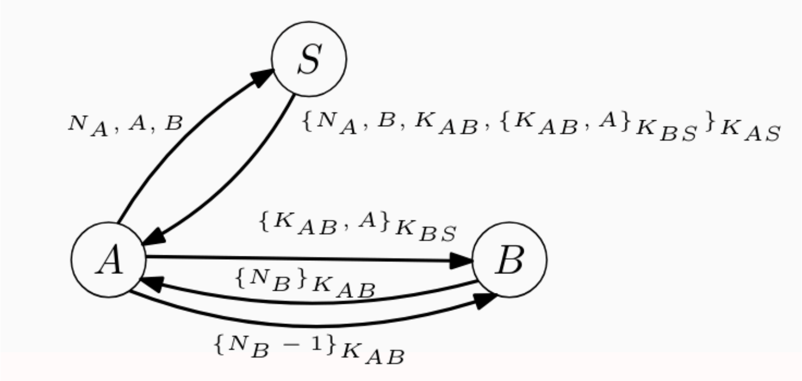


Figure 19: Needham-Schroeder Protocol

The encryption is denoted as $\{M\}_k$: a message M and a key k.

It is a symmetric key exchange algorithm that creates a session-key for two participants A, B, named K_{AB}.

The session key is also encrypted during the communication thanks to K_{AS} and K_{BS}. So you're able to look the message only if you also know the key: this is called expression, we do not talk about just binary strings.

There are some issues for this protocol that identifies it as non-secure, but they're not about the cryptographic primitives used. They're at logical level of abstraction.

1. An attacker can hold a previous version of K_{AS} so to have a replay attack.
   - B generates a valid N_B but the attacker can decrypt the produced message and produces $N_B - 1$.
   - The attacker impersonates A.
2. If we do not use B as part of the step 2 (the response $S \to A$), an attacker could intercepts the message at step 1 and edit the "receiver" as C.
   - A would not be able to recognize any impersonates.

In the symbolic model we have some differences:

	Computational	**Formal**
Messages	Binary strings	Expressions
Adversaries	Efficient Algorithms	Arbitrary Processes
Attacks	Non Negligible Probability Event	Possible Event

If you have a chance to break the scheme that is a real attack, we do not talk about probabilities.

9.2. Expressions

An expression used above is the $\{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$. We see nested components into the main expression component. Saying that we can't see whole the expression as a simple string of bits.

For $\{K_{AB}, A\}$ an adversary could be able to know the expression but it should also know the K_{BS} to be able to understand what it's looking at.

It is possible to parameterise which cryptographic primitives are available.

A difference for computational models but common to all symbolic models is the simplicity:
- Protocols' security can be proved without assumptions.
- Given a protocol, deciding whether an adversary exists is a problem that can also be faced with automatic techniques.
- As a consequence, the symbolic model can form the basis for model-checking, semi-automated theorem proving, logic programming.
- All this has led to the design of concrete tools such as `ProVerif` or `Tamerin`.

9.3. ProVerif

A tool which analyses a protocol. You can define your own cryptographic primitives in an abstract level.

The output is $\{Y, N, ?\}$. Y means "the verification holds"; N means "no"; ? means "who knows".

```
free c: channel .
free s: bitstring [ private ].

query attacker (s).

process
out (c, s);
0
```

- `channel` is a predefined type
- `0` means `EOF`.
- `bitstring` is not a binary digit string, but it can be anything else.
- `query` checks if the attacker knows the parameter `s`.
- an output `RESULT not attacker(s[]) is false.` means the attacker knows s.

```
free c: channel .
free s: bitstring [ private ].
free p: bitstring [ private ].
query attacker (s).

process
out (c, p);
0
```

- If we set `p` as output, we have `RESULT not attacker(s[]) is true`.

```
free c: channel .
free s: bitstring .
free p: bitstring [ private ].
query attacker (s).

process
out (c, p);
0
```

- `s` is not private, we have `RESULT not attacker(s[]) is false`.

9.3.1. A real example

```
type key .

fun encrypt (bitstring, key): bitstring .
fun decrypt (bitstring, key): bitstring .
equation forall x: bitstring, y: key; decrypt (encrypt (x, y), y) = x .
equation forall x: bitstring, y: key; encrypt (decrypt (x, y), y) = x .

free c: channel .
free k: key [ private ].
free s: bitstring [ private ].

query attacker (s).

let processA =
out (c, encrypt (s, k)).

let processB =
in (c, x: bitstring);
let n = decrypt (x, k) in
0.

process
(! processA) | (! processB)
```

- The only reason why key is not `bitstring` is an intuitive annotation (it is a string ofc).
- An adversary asks for equations.
- `|` stands for "parallel".
- `processA` and `processB` is named macros or functions: they're parameters in input.
- The only way to know `s` is to apply the first equation but `key` is private.

With an edit

```
let processB =
in (c, x : bitstring);
```

```
let n = decrypt (x, k) in
out (c, k).
```

- `processB` is stupid because it returns the decrypted value. The attacker sees the key.

A bad event does not happen or whenever an event happen another one has happened previously.

The keyword `new` allows for the definition of local names, which are invisible from the outside.

9.4. Multiset rewriting

In the symbolic model key, message, nonce, cipher can be separated and not considered all as strings.

- Different types (**sort**).
- A **function symbol** is a function which has sort as parameters (and that outputs a single tuple).

$$\text{enc} : \text{key} \times \text{msg} \mapsto \text{cipher}$$

$$\text{dec} : \text{key} \times \text{cipher} \mapsto \text{msg}$$

- **Predicates**: names of properties of tuples of elements.

$$\textsc{Knowledge} : \text{cipher}$$

$$\textsc{KeyPair} : \text{pubkey} \times \text{privkey}$$

- **Terms** are expressions built from variables and function symbols.

$$k : \text{key}, m : \text{msg} \vdash \text{enc}(k, m) : \text{cipher}$$

- **Facts** are predicates applied to expressions with the right sort (type). They use something that are supposed to be true.

$$k : \text{key}, m : \text{msg} \vdash \textsc{Knowledge}\ (\text{enc}(k, m))$$

- **State** is a finite multiset of facts, a set in which each element can occur more than once. Given n (not necessarily distinct) facts $A_1, ..., A_n$, the

multiset which contains them is indicated simply as $A_1, ..., A_n$.

$$\textsc{Knowledge}\ (\text{enc}(k, m)), \textsc{Knowledge}\ (k)$$

- **Signature** is a set of sorts and a set of function symbols and predicates.
- **Rules** are the dynamic evolution of the underlying state of the form

$$A_1, ..., A_n \to \exists x_1, ... \exists x_m . B_1, ..., B_k$$

where $n, m, k \geq 0$, while the A_i-s and B_j-s are facts.

$$\textsc{Knowledge}\ (\text{enc}(k, m)), \textsc{Knowledge}\ (k) \to \textsc{Knowledge}\ (m)$$

- A signature and a set of rules over it forms a **theory**.
- A **trace** is a sequence of multi fact.

9.4.1. Theory of FA

We now define:

- Sorts: state, symb, string.
- Function symbols, with everything finite:

$$\text{cons} : \text{symb} \times \text{string} \to \text{string}$$

$$a_1, ..., a_m : \text{symb}$$

$$q_1, ..., q_n : \text{state}$$

$$\text{nil} : \text{string}$$

- Predicates:

$$\textsc{CurrentState} : \text{state}$$

$$\textsc{InputLeft} : \text{string}$$

- Rules:

$$\textsc{CurrentState}\ (q_i), \textsc{InputLeft}\ \left(\text{cons}\left(a_j, x\right)\right) \to \textsc{CurrentState}\ (q_k), \textsc{InputLeft}\ (x)$$

$\delta\left(q_i, a_j\right) = a_j$ is the transaction function of the FA.

There's no only one theory for every finite automata.

The ∃ quantifier does not exist: this would make the task analyser harder.

9.4.2. Turing Machine

In this case we need the ∃. Now, let's add the sort cells and predicates

$\textsc{Content}$: cell × symb

$\textsc{Adjacency}$: cell × cell

if one cell is adjacent to another, it means the second is next to first, just like an order.

$$\textsc{Adjacency}\ (x, c_{eot}) \to \exists y.\textsc{Adjacency}\ (x, y), \textsc{Content}\ (y, \text{blank}), \textsc{Adjacency}\ (y, c_{eot})$$

It is not modeling the computation but the growing of the model.

9.4.3. Safety problems

The problem of "bad" situations never happen is actually a safety problem.

Given a theory, a set of initial facts X and set of bad facts Y (we do not want to be actual in practice), a MSR safety problem determines whether there exist a trace leading from a fact in X to a fact in Y, namely a trace in the form

$$S_1 \to S_2 \to ... \to S_n$$

where $S_1 \in X$ and $S_n \in Y$.

A particular safety problem is an halting problem, so it is quite hard.

Theorem The MSR safety problem is undecidable.

9.4.4. Protocols as theories

Finite automata and Turing machines are sequential models of computation.

For every agent X and for every phase $i \in \mathbb{N}$ in the execution of the protocol, there is a predicate X_i capturing the fact that X is in phase i, and that it knows some data, seen as a parameter to X_i.

The exchange of data between the parties is mediated by the network, and this is captured by a predicate N_i, this way paving the way to the modeling of attackers.

Example

$A \to B : N_A$

$B \to A : N_A, N_B$

$A \to B : N_B$

we can model this model using a single sort n which stands for nonce.

Predicates, that are some phases:

$A_0 : 1 \qquad A_1 : n \qquad A_2 : n \times n$

- A_0 is the first stance; A_1 is it produces a nonce; A_2 is the final case.

$B_0 : 1 \qquad B_1 : n \times n \qquad B_2 : n \times n$

- B_1 is the case when A sends to it the nonce.

$N_1 : n \qquad N_2 : n \times n \qquad N_3 : n$

The rules are

$A_0 \to \exists x. A_1(x), N_1 x$

$B_0, N_1(x) \to \exists y. B_1(x, y), N_2(x, y)$

$A_1(x), N_2(x, y) \to A_2(x, y), N_3(y)$

$B_1(x, y), N_3(y) \to B_2(x, y)$

9.4.5. Modeling the attacker

In the Dolev-Yao model an attacker can:

- Read any message, preventing it to reach its destination.
- Decompose a message into parts and remember them (including decrypting a ciphertext for which it has obtained the key).
- Generate fresh data.
- Compose a new message from known data and send it to the network.

This can be modeled by endowing the theory with:

- A predicate D capturing what the attacker has observed.
- A predicate M modeling the intruder's memory.
- A predicate C which serves to model new messages the adversary has crafted, and which could possibly be sent.

It is convenient that the aforementioned (unary) predicates are on a sort m of which n is a subsort, and that a binary function symbol $\langle\cdot,\cdot\rangle$ on m is available.

The rules are

$N_1(x) \to D(x)$

$N_2(x, y) \to D(\langle x, y\rangle)$

$D(\langle x, y\rangle) \to D(x), D(y)$ [this is decomposed]

$D(x) \to M(x)$

$M(x) \to C(x), M(x)$

$C(x) \to N_1(x)$

$C(x), C(y) \to C(\langle x, y\rangle)$

$C(\langle x, y\rangle) \to N_2(x, y)$

$\to \exists x.M(x)$ [it means you can create anything]

The memory is persistent, so it can be used to reconstruct the messages on the network.

There is a trace starting in the A_0, B_0 and ending in a state in which A and B concluded the protocol without sharing the same data.

9.4.6. A toy protocol

$$A_0, B_0 \to A_1(n_A), N_1(n_A), B_0$$

$$\to A_1(n_A), B_1(n_A, n_B), N_2(n_A, n_B)$$

$$\to A_1(n_A), B_1(n_A, n_B), D(\langle n_A, n_B\rangle)$$

- in a pessimistic way to see the things, the attacker has access to the whole network.

$$\to A_1(n_A), B_1(n_A, n_B), D(n_A), D(n_B)$$

$$\xrightarrow{2} A_1(n_A), B_1(n_A, n_B), M(n_A), M(n_B)$$

- the attacker wants to create a new nonce.

$$\to A_1(n_A), B_1(n_A, n_B), M(n_A), M(n_B), M(n_C)$$

$$\xrightarrow{2} A_1(n_A), B_1(n_A, n_B), M(n_A), M(n_B), M(n_C), C(n_A), C(n_C)$$

$$\to A_1(n_A), B_1(n_A, n_B), M(n_A), M(n_B), M(n_C), C(\langle n_A, n_C\rangle)$$

$$\to A_1(n_A), B_1(n_A, n_B), M(n_A), M(n_B), M(n_C), N_2(n_A, n_C)$$

$$\to A_2(n_A, n_C), B_1(n_A, n_B), M(n_A), M(n_B), M(n_C), N_3(n_C)$$

$$\to A_2(n_A, n_C), B_1(n_A, n_B), M(n_A), M(n_B), M(n_C), D(n_C)$$

$$\to A_2(n_A, n_C), B_1(n_A, n_B), M(n_A), M(n_B), M(n_C), C(n_B), D(n_C)$$

$$\to A_2(n_A, n_C), B_1(n_A, n_B), M(n_A), M(n_B), M(n_C), N_3(n_B), D(n_C)$$

$$\rightarrow \underbrace{A_2(n_A, n_C), B_2(n_A, n_B)}_{\substack{\text{this is a final configuration in which} \\ A \text{ and } B \\ \text{do not have the same view}}}, M(n_A), M(n_B), M(n_C), D(n_C)$$

9.5. Symbolic model through expressions

A trivial symbolic model starting with the set $\mathbb{B} = \{0, 1\}$ of booleans and set $\mathbb{K}$ of key symbols (they're not binary strings, they are atomic).

The expressions we're considering are

$$M, N \Coloneqq K \mid i \mid \langle M, N \rangle, \{M\}_K$$

where $i \in \mathbb{B}$ and $K \in \mathbb{K}$. If you know the encryption $\{M\}_K$ you don't know the component M.

An implication is expressed by $M \vdash N$, that means M implies N.

$$\frac{}{M \vdash 0}$$

$$\frac{}{M \vdash 1}$$

$$\frac{}{M \vdash M}$$

$$\frac{M \vdash N \qquad M \vdash L}{M \vdash (N, L)}$$

$$\frac{M \vdash \langle N, L \rangle}{M \vdash N}$$

$$\frac{M \vdash \langle N, L \rangle}{M \vdash L}$$

a pair is different than $\{N\}_K$ because you do not need to know K.

$$\frac{M \vdash N \qquad M \vdash K}{M \vdash \{N\}_K}$$

$$\frac{M \vdash \{N\}_K \qquad M \vdash K}{M \vdash N}$$

For instance, $\left(\left\{\{K_1\}_{K_2}\right\}_{K_3}, K_3\right) \vdash K_3$.

9.5.1. Equivalances

If E is an expression set, $\{M \mid \exists N \in \mathrm{E}. N \vdash M\}$ is what the adversary can compute from E.

At each step, the adversary may compute any expression among those in the set and use it to construct an attack.

Two expressions are considered equivalent if they are indistinguishable with respect to an adversary whose task is to "separate" them.

For example, $\left(0, \{0\}_{K_1}\right)$ and $\left(0, \{1\}_{K_2}\right)$ are equivalent because the second part of both messages is the same, even if the message and the keys are different.

9.5.2. Patterns

The way an adversary "sees" an expression is captured by the notion of pattern.

$$P, Q ::= K \mid i \mid \langle P, Q \rangle \mid \{P\}_K \mid \square$$

The pattern to which an expression corresponds, depends on the set of keys $\mathcal{T}$ available to the the adversary. We can read those as "the view of a something is somethat".

$p(K, \mathcal{T}) = K$

$p(i, \mathcal{T}) = i$

$p(\langle M, N \rangle, \mathcal{T}) = \langle p(M, \mathcal{T}), p(N, \mathcal{T}) \rangle$

$p(\{M\}_K, \mathcal{T}) = \begin{cases} \{p(M, \mathcal{T})\}_K & \text{if } K \in \mathcal{T} \\ \square & \text{otherwise} \end{cases}$

An adversary sees an expression M as

$$\text{pattern}(M) = p(M, \{K \in \mathbb{K} \mid M \vdash K\})$$

Two expressions M and N are equivalent iff $\text{pattern}(M) = \text{pattern}(N)$.

An equivalence is sait to be weak if

$$M \cong N \Longleftrightarrow M \equiv N\sigma$$

with σ bijetion on $\mathbb{K}$.

For instance, we two keys $K_1 \cong K_2$ they are different but should they be not equivalent? Not really. Indeed, $\{0\}_K \cong \{K\}_K$ but $(K, \{0\}_K) \ncong (K, \{1\}_K)$. The things equivalent are causing the renaming of the values, like the α equivalence of the λ calculcus.

9.6. Relating formal and computational models

We want to define when two expressions are equivalent accoring the computational model. An example using $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ we need to define:

- Security parameter n.
- Every $k \in \mathbb{K}$ that occurs in M is what we get from $\mathsf{Gen}(1^n)$.
- 0 and 1 are strings from $\mathbb{B}$.
- $\langle N, L \rangle$ in M are binary strings.
- A ciphertext $\{N\}_K$ in M has got by the invoking of Enc.

The famility of distributions corresponding to an exepression is the semantic of the expression parametrized on the used encryption scheme: $[\![M]\!]_\Pi$.

When two expression are equivalent according to the computational model?

Definition Two families of distributions $\mathcal{D} = \{D_n\}_{n \in N}$ and $\mathcal{E} = \{E_n\}_{n \in N}$ are called computationally indistinguishable iff for each PPT adversary A there exists a negligible function $\varepsilon \in \mathcal{NGL}$ such that

$$|\Pr(A(1^n, D_n) = 1) - \Pr(A(1^n, E_n) = 1)| \leq \varepsilon(n)$$

and we write $\mathcal{D} \sim \mathcal{E}$, so $\mathcal{D}$ is indistinguishable computable to $\mathcal{E}$ (said equivalent).

The expression above is similar to the expression seen for PRG.

An expression M is said to be *acyclic* iff for every subexpression $\{N\}_K$ of M, the key K does not occur in N.

Theorem (Abadi&Rogaway)

If Π is secure (in CPA) and M, N are acyclic, then $M \cong N$ implies $[\![M]\!]_\Pi \sim [\![N]\!]_\Pi$.

So, they are computational secure.