2021-03-04 theory

March 2021

1.4.1 The operation ⊠

For a given $a, b \in 0, 1^5$ the operation ⊠ is the multiplication in base 2 disregarding bits of positions $> n$ (in the example below $n = 5$, just for teaching purposes). For example, if a = [01010] and b = [10011] then we can compute a ⊠ b by hand as:

					0	1	0	1	0
				⊠	1	0	0	1	1
					0	1	0	1	0
				0	1	0	1	0	
0	1	0	1	0					
					1	1	1	1	0

Actually, if a,b are regarded as integers in base 2, i.e. a = 10,b = 19, then the performed operation is: a ⊠ b = (a · b)$(mod 2^n)$. In the current example:

$$10 \boxtimes 19 = 30$$

because $10 \cdot 19 = 190 = 30(mod 32)$ (in python (10*19)%32 gives 30).
Exercise 1.4.3: Show that $a \boxtimes b = (a \cdot b)(mod 2^5)$. See the previous example.

Exercise 1.4.3

Let $a, b \in \{0, 1\}^5$. Show that $a \boxtimes b = (a \cdot b)(mod\ 2^5)$.

Let's start with an *observation* (1): dividing a number by 2^n is the same as shifting the binary number to the right by n positions, while multiplying a number by 2^n is the same as shifting the binary number to the left by n positions. For example, if $x = 11 = [1011]$, then:

$$\frac{x}{2^2} = \frac{11}{4} = [1011] \gg 2 = [0010] = 2$$

And also:

$$x \cdot 2^2 = 11 \cdot 4 = [1011] \ll 2 = [101100] = 44$$

Now let's write the product $a \cdot b$ in its generic binary form. It's easy to see that this number takes at most 9 bits.

$$a \cdot b = [b_8\ b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0]$$

Where $b_8 \dots b_0$ are the bits of the binary string. When we perform the operation $a \boxtimes b$ we simply cut the bits from b_5 to b_8:

$$a \boxtimes b = [b_4\ b_3\ b_2\ b_1\ b_0]$$

On the other hand, when we write $(a \cdot b)(mod\ 2^5)$ we are taking the remainder of the division between $(a \cdot b)$ and 2^5. Then we can write:

$$(a \cdot b)(mod\ 2^5) = (a \cdot b) - q \cdot 2^5$$

Where q is the quotient of the integer division between $(a \cdot b)$ and 2^5. Since (1) we can write q as:

$$q = \frac{a \cdot b}{2^5} = (a \cdot b) \gg 5\ =\ [b_8\ b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0] \gg 5\ =$$

$$= [0\ 0\ 0\ 0\ b_8\ b_7\ b_6\ b_5\ b_4]$$

Moreover, still since (1), we can write $q \cdot 2^5$ as:

$$[0\ 0\ 0\ 0\ b_8\ b_7\ b_6\ b_5\ b_4] \ll 5 = [b_8\ b_7\ b_6\ b_5\ b_4\ 0\ 0\ 0\ 0\ 0]$$

Then:

$$(a \cdot b)\ (mod 2^5) = (a \cdot b) - 2^5 =$$

$$[b_8\ b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0]\ -\ [b_8\ b_7\ b_6\ b_5\ b_4\ 0\ 0\ 0\ 0\ 0] =$$

$$= [b_4\ b_3\ b_2\ b_1\ b_0]$$

But this is exactly $a \boxtimes b$.

q.e.d.

Exercise 1.4.4: Find $x \in 0, 1^5$ such that $19 \boxtimes x = 1$.
As we can see from the previous exercise, we'll obtain: $19 \boxtimes x = (19 \cdot x)(mod 2^5)$, then: $(19 \cdot x)(mod 2^5) = 1$. So, a naive method of finding a modular inverse for A (mod C) is:

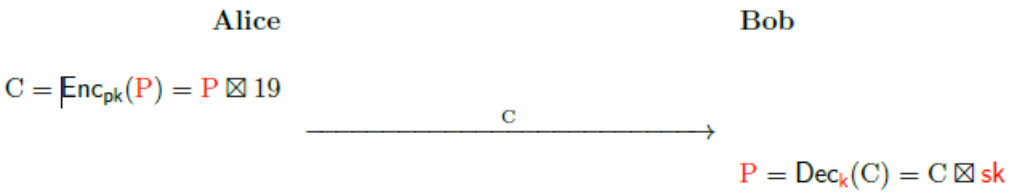
- step 1. Calculate A * B mod C for B values 0 through C-1
- step 2. The modular inverse of A mod C is the B value that makes A * B mod C = 1

Note that the term B mod C can only have an integer value 0 through C-1, so testing larger values for B is redundant.

In this exercise we perform these calculations below:
- $19 \cdot 0 = 0 \ (mod\ 32)$
- $19 \cdot 1 = 19 \ (mod\ 32)$
- $19 \cdot 2 = 6 \ (mod\ 32)$
- $19 \cdot 3 = 25 \ (mod\ 32)$
- $19 \cdot 4 = 12 \ (mod\ 32)$
- $\cdots$
- $19 \cdot 27 = 1 \ (mod\ 32) \longleftarrow$ inverse found!

1.4.2 A baby example of asymmetric cipher

Bob's pair (sk, pk): the public key is pk = 19 and the secret key sk is the x such that $19 \boxtimes x = 1$. Here is how Alice use Bob's public key to encipher $P \in 0, 1^5$:



Exercise 1.4.5: *Is it possible to find Bob's secret key sk in the case above?*
There are two possible solutions to this exercise:

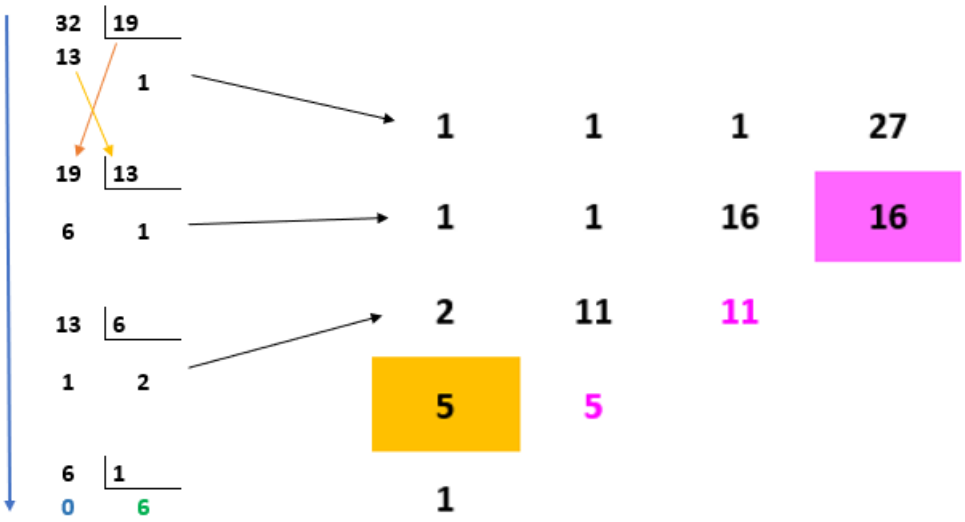
- By means of a brute force loop on python
- Using the Kuttaka algorithm

The Kuttaka algorithm

Hypotesis: the pair (sk, pk) is calculated by: pk ⊠ sk = 1, so that the sk is the inverse of the pk.
The Kuttaka algorithm allow to find the secret key (sk) with calculations made by hand, starting from a public key (pk), by computing the inverse of numbers.
Example of Kuttaka algorithm implementation:
Bob's pk is [10011] = 19. The used strategy is:

$$19 \cdot \equiv 1(mod 32) \leftrightarrow 19 \boxtimes x \equiv 1(mod 32)$$

Then, multiple divisions are needed, starting from 2^n/pk, until a remainder with value 0 is reached:



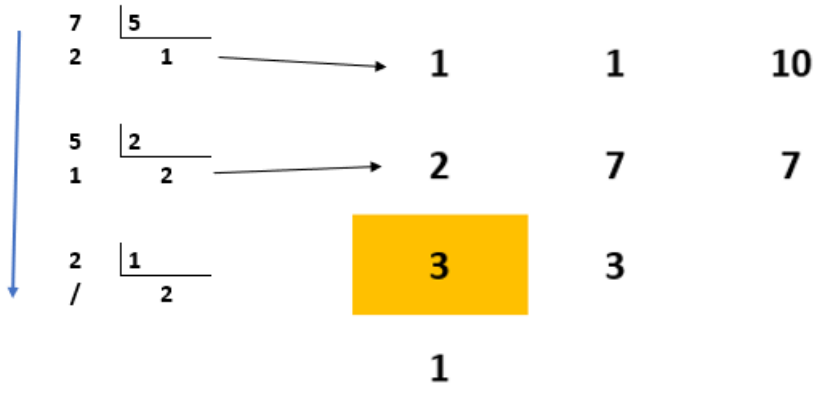
The previous table is created with these rules:

1. The first column has M = X + 1 cells, where X is the number of division performed;
2. The first values of the first column are copied from the results of the divisions;
3. The last values is always 1;
4. The second-to-last value of the first column is computed as follows:
   - if M is even, then the value is computed by subtracting 1 to the result of the last division
   - if M is odd, then the value is computed by adding 1 to the result of the last division
5. The second-to-last value of each other column is computed as the sum of the multiplication of the previous second-to-last element with the element above it (11·1), plus the last values of previous column (5). In the example: $16 = 11 \cdot 1 + 5$.
6. Other numbers are just copied to the next column
7. The final result is contained in the top-right corner of the table

In order to verify the final result it is possible to use python:

$$19^{-1} \equiv 27(mod32)$$

Another example of Kuttaka algorithm: $\frac{3}{5}$ in $\mathbb{Z}_7$:



1.5 A baby Galois ⊗-cipher

Following Evariste Galois we introduce a new multiplication $\otimes$ between strings $a, b \in 0, 1^5$:

$$a \otimes b = c$$

Be careful to not confuse $\otimes$ with a XOR ($\oplus$).

Let's take a Galois modular function G(x) and two polynomials a(x) and b(x), coming from $a = [a_4a_3a_2a_1a_0]$ and $b = [b_4b_3b_2b_1b_0]$, respectively:

$$a(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$$
$$b(x) = b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$$

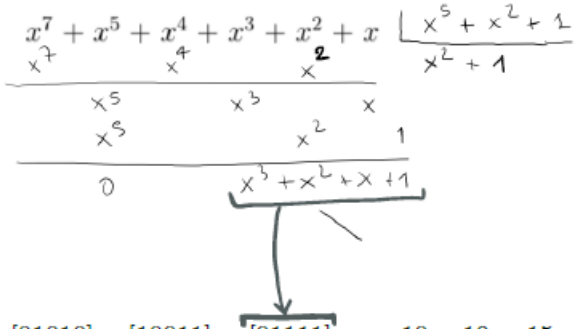
The Galois multiplication consist in compute $(a(x) \times b(x))modG(x)$, with a remainder $c(x)$.
As an explicit example let us compute [01010] $\otimes$ [10011]. First of all we compute

$$(0x^4 + 1x^3 + 0x^2 + 1x + 0) \times (1x^4 + 0x^3 + 0x^2 + 1x + 1)\,.$$

Namely:

$$\begin{array}{cccccccccc}
 & & & & & 0x^4 & 1x^3 & 0x^2 & 1x & 0 \\
 & & & & \times & 1x^4 & 0x^3 & 0x^2 & 1x & 1 \\
\hline
 & & & & & 0x^4 & 1x^3 & 0x^2 & 1x & 0 \\
 & & & & 0x^5 & 1x^4 & 0x^3 & 1x^2 & 0x & \\
0x^8 & 1x^7 & 0x^6 & 1x^5 & 0x^4 & & & & & \\
\hline
0x^8 & 1x^7 & 0x^6 & 1x^5 & 1x^4 & 1x^3 & 1x^2 & 1x & 0 &
\end{array}$$

and we get $x^7 + x^5 + x^4 + x^3 + x^2 + x$. Now its remainder when divided by $x^5 + x^2 + 1$.



So

$$[01010] \otimes [10011] = [01111]\,, \text{ or } 10 \otimes 19 = 15$$

In the end, the result is: [01111] $(mod G(x))$.

Bob's pair (sk, pk): the public key is pk $= 19$ and the secret key sk is the x such that $19 \otimes x = 1$. Here is how Alice use Bob's public key to encipher $\mathrm{P} \in \{0,1\}^5$:

Alice		**Bob**
$\mathrm{C} = \mathsf{Enc}_{\mathsf{pk}}(\mathrm{P}) = \mathrm{P} \otimes 19$		
	$\xrightarrow{\quad \mathrm{C} \quad}$	
		$\mathrm{P} = \mathsf{Dec}_{\mathsf{k}}(\mathrm{C}) = \mathrm{C} \otimes \mathsf{sk}$

Exercise 1.5.2: Can you find Bob's secret key sk in the previous example? The secret key sk is the x such that $19 \otimes x = 1$. For a given finite field $GF(2^m)$ and the corresponding irreducible reduction polynomial $G(x)$, the inverse A^-1 of a nonzero element $A \in GF(x)$ is defined as:

$$A^-1(x) \cdot A(x) = 1 \; mod \; G(x)$$

In order to find the Galois multiplicative inverse of a value, it is possible to use a multiplicative inverse table. For small fields lookup tables with the precomputed inverses of all fields elements are often used. As an example, we can consider the table below for inverse values in $GF(2^8)$:

		Y														
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0	00	01	8D	F6	CB	52	7B	D1	E8	4F	29	C0	B0	E1	E5	C7
	1	74	B4	AA	4B	99	2B	60	5F	58	3F	FD	CC	FF	40	EE	B2
	2	3A	6E	5A	F1	55	4D	A8	C9	C1	0A	98	15	30	44	A2	C2
	3	2C	45	92	6C	F3	39	66	42	F2	35	20	6F	77	BB	59	19
	4	1D	FE	37	67	2D	31	F5	69	A7	64	AB	13	54	25	E9	09
	5	ED	5C	05	CA	4C	24	87	BF	18	3E	22	F0	51	EC	61	17
	6	16	5E	AF	D3	49	A6	36	43	F4	47	91	DF	33	93	21	3B
	7	79	B7	97	85	10	B5	BA	3C	B6	70	D0	06	A1	FA	81	82
X	8	83	7E	7F	80	96	73	BE	56	9B	9E	95	D9	F7	02	B9	A4
	9	DE	6A	32	6D	D8	8A	84	72	2A	14	9F	88	F9	DC	89	9A
	A	FB	7C	2E	C3	8F	B8	65	48	26	C8	12	4A	CE	E7	D2	62
	B	0C	E0	1F	EF	11	75	78	71	A5	8E	76	3D	BD	BC	86	57
	C	0B	28	2F	A3	DA	D4	E4	0F	A9	27	53	04	1B	FC	AC	E6
	D	7A	07	AE	63	C5	DB	E2	EA	94	8B	C4	D5	9D	F8	90	6B
	E	B1	0D	D6	EB	C6	0E	CF	AD	08	4E	D7	E3	5D	50	1E	B3
	F	5B	23	38	34	68	46	03	8C	DD	9C	7D	A0	CD	1A	41	1C

For example, from the table, we get that the inverse of

$$x^7 + x^6 + x = [11000010]_2 = C2_{hex}$$

is given by the element in row $C(12_{10})$, column 2:

$$2F_{hex} = [00101111]_2 = x^5 + x^3 + x^2 + x + 1.$$

This can be verified by multiplication:

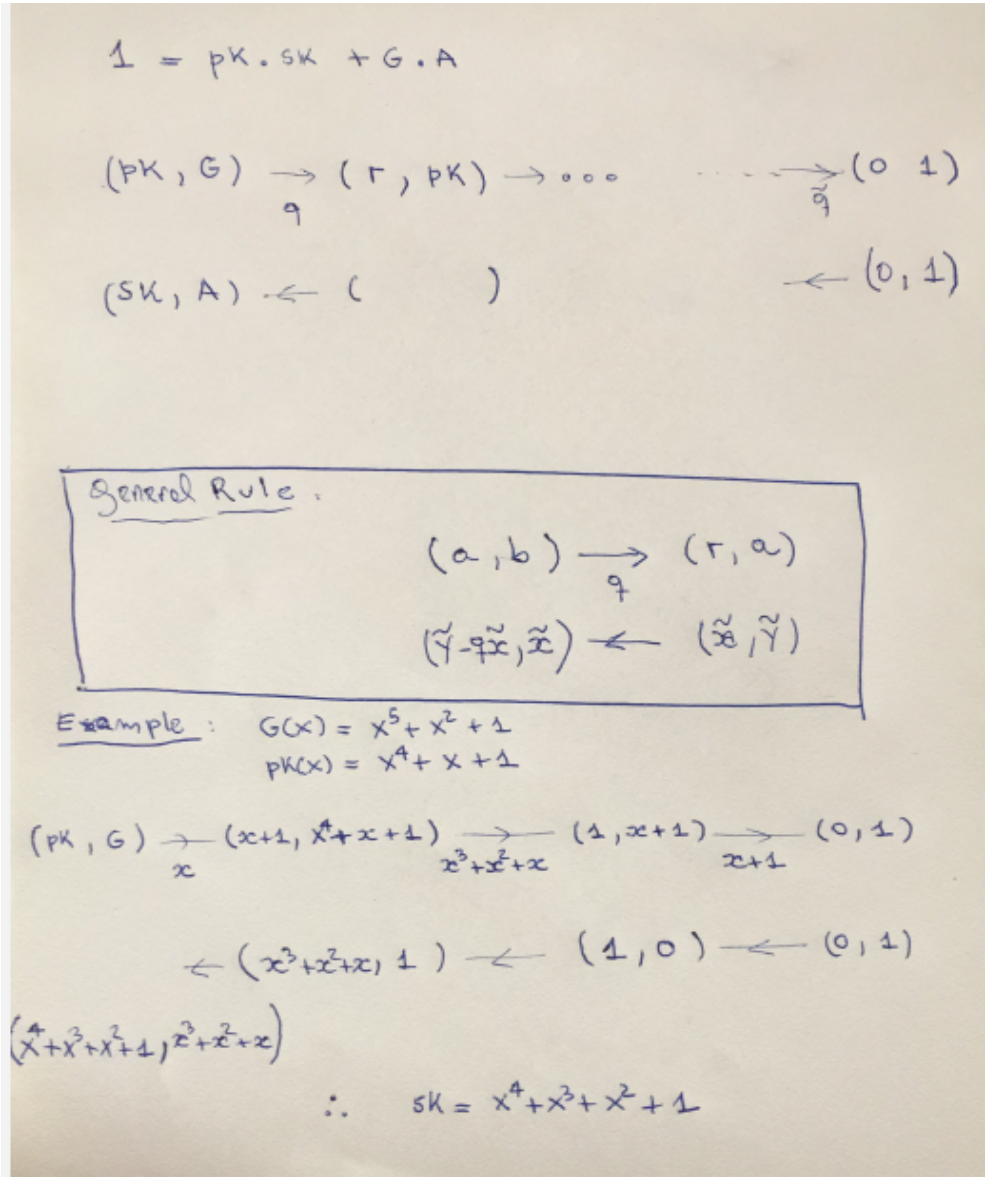
$$(x^7 + x^6 + x) \cdot (x^5 + x^3 + x^2 + x + 1) \equiv 1 \; mod \; G(x).$$

As an alternative to using lookup tables, one can also explicitly compute inverses. The main algorithm for computing multiplicative inverses is the Extended Euclidean algorithm:

Extended Euclidean Algorithm (EEA)
Input: positive integers r_0 and r_1 with $r_0 > r_1$
Output: $\gcd(r_0, r_1)$, as well as s and t such that $\gcd(r_0, r_1) = s \cdot r_0 + t \cdot r_1$.
Initialization:
$s_0 = 1 \quad t_0 = 0$
$s_1 = 0 \quad t_1 = 1$
$i = 1$
Algorithm:

1 DO
1.1 $i = i + 1$
1.2 $r_i = r_{i-2} \bmod r_{i-1}$
1.3 $q_{i-1} = (r_{i-2} - r_i)/r_{i-1}$
1.4 $s_i = s_{i-2} - q_{i-1} \cdot s_{i-1}$
1.5 $t_i = t_{i-2} - q_{i-1} \cdot t_{i-1}$
WHILE $r_i \neq 0$
2 RETURN
$\gcd(r_0, r_1) = r_{i-1}$
$s = s_{i-1}$
$t = t_{i-1}$

For example, the exercise we were doing before can be solved as follow:

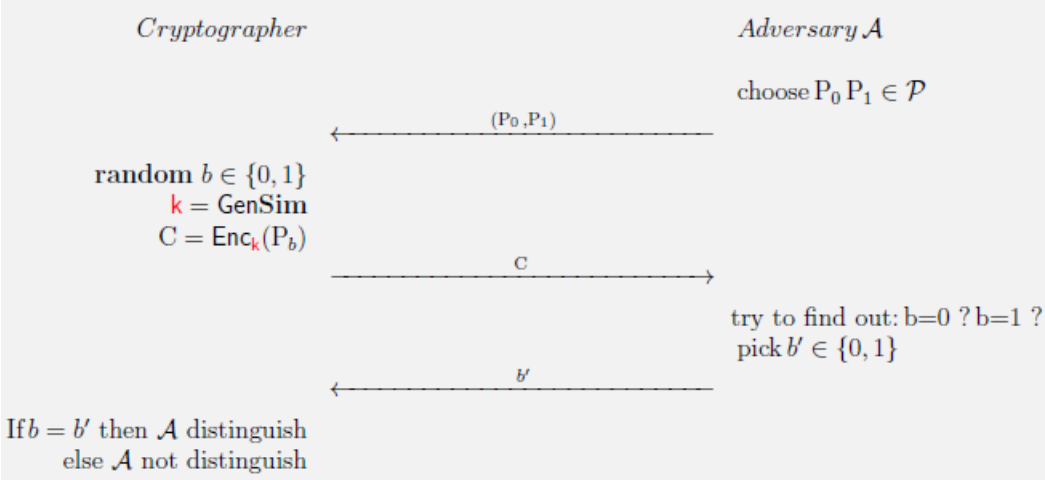


1.6 Symmetric Cryptography

1.6.1 IND Indistinguishability experiment

Ciphertext indistinguishability is a property of many encryption schemes. Intuitively, if a cryptosystem possesses the property of indistinguishability, then an adversary will be unable to distinguish pairs of ciphertexts based on the message they encrypt. The property of indistinguishability under chosen plain-

text attack is considered a basic requirement for most provably secure public key cryptosystems, though some schemes also provide indistinguishability under chosen ciphertext attack and adaptive chosen ciphertext attack.
IND is a theoretical experiment used to evaluate indistinguishability of a cryptosystem. In this experiment there is a cryptoanalyst (adversary) that sends us two different plaintexts. We, acting as a cryptographer, have to randomly choose one of the two using a *coin mechanism* (like heads and tails game) and encrypt it. If the cryptoanalyst will be unable to distinguish our random choice (head or tail), then the cryptosystem described by this specific experiment is considered secure regarding the indistinguishability.
The experiment is repeated several times and a probability of the chance to distinguish the random choice is calculated (probability $p \in [0, 1]$).



The symmetric cryptosystem Π = (Gen, Enc, Dec) is defined as **IND-secure** if and only if no adversary $\mathscr{A}$ can distinguish with probability better than 1/2.

1.7 The key distribution problem

The goal is for two users, A and B, to securely exchange a key over an insecure channel. This key is then used by both users in normal cryptosystem for both enciphering and deciphering.

Key exhange protocol: is a protocol, i.e. a set of instructions for the parties (Alice and Bob), that starting from a security parameter n allows them to compute keys k_A and k_B. The correctness requirement is that

$$k_A = k_B$$

so we can speak simply of the key $k = k_A = k_B$ as the exchanged key. The security of the exchange protocol is related to the following **key-exchange ex-**

periment:

1. Two parties holding 1^n execute protocol $\prod$. This execution of the protocol results in a transcript *trans* containing all the messages sent by the parties, and a key k that is oupput by each of the parties.
2. A random bit b is chosen. If b = 0 then choose k' randomly too, otherwise, if b = 1 set k' = k.
3. *trans* and k' are given to the adversary, which generates b'
4. Experiment output is 1 if b' = b (success), otherwise it is 0.

A protocol $\prod$ is considered **$\prod$-secure** if no adversary can succeed with probability better than 1/2.

Active adversaries. So far we have considered only the case of an eavesdropping adversary. Although eavesdropping attacks are by far the most common (as they are so easy to carry out), they are by no means the only possible attack. *Active* attacks, in which the adversary sends messages of its own to one or both of the parties are also a concern, and any protocol used in practice must be resilient to active attacks as well. When considering active attacks, it is useful to distinguish, informally, between *impersonation* attacks where only one of the honest parties is executing the protocol and the adversary impersonates the other party, and *man-in-the-middle* attacks where both honest parties are executing the protocol and the adversary is intercepting and modifying messages being sent from one party to the other.

We will not define security against either class of attacks, as such a definition is rather involved and also cannot be achieved without the parties sharing *some* information in advance. Nevertheless, it is worth remarking that the Diffie-Hellman protocol is *completely insecure* against man-in-the-middle attacks. In fact, a man-in-the-middle adversary can act in such a way that Alice and Bob terminate the protocol with different keys k_A and k_B that are both known to the adversary, yet neither Alice nor Bob can detect that any attack was carried out. We leave the details of this attack as an exercise.

The fact that the Diffie-Hellman protocol is not resilient to man-in-the-middle attacks does not detract in any way from its importance. The Diffie-Hellman protocol served as the first demonstration that asymmetric techniques (and number-theoretic problems) could be used to alleviate the problems of key distribution in cryptography. Furthermore, extensions of the Diffie-Hellman protocol can be shown to prevent man-in-the-middle attacks, and such protocols are widely used today.

1.8 Public Key Cryptography

Such cryptosystem consists of three algorithms (Gen , Enc , Dec).
Gen generate a pair (sk,pk) of secret key sk and public key pk.

1) The algorithms Gen, Enc e Dec must be computationally feasible,
2) The secret key sk should be computationally infeasible to compute from the public key pk.
Property 2) allows the publication of pk in public web pages. The security of the public key cryptosystem is related to the following **eavesdropping indistinguishability experiment**:

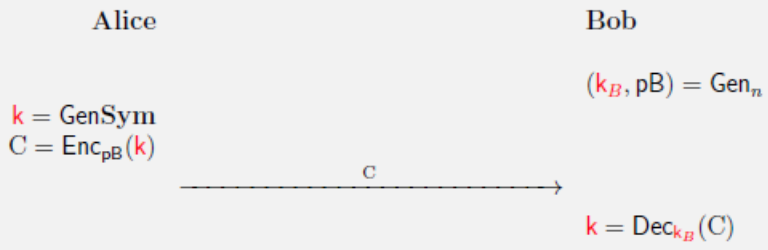
1. Gen(1^n) is ran to obtain keys (pk, sk)
2. pk is given to the adversary, and it will output a pair of messages m_0, m_1 of the same length (these messages must be in the plaintext space $\mathscr{P}$ associated with pk)
3. A random bit b is chosen. Then a challenge ciphertext c is generated by Enc(m_b) and it is given to the adversary.
4. The adversary generates a bit b'
5. Experiment output is 1 if b' = b (success), otherwise it is 0.

A protocol $\prod$ is considered $\prod$**-secure** if no adversary can succeed with probability better than 1/2.

1.8.1 From PKC to PKDS

Usually, the pair (pk,sk) is used in order to generate a temporary symmetric key when the two parties need to encipher big plaintexts (PKDS, Public Key Distribution System):

Alice and Bob are users of a **PKC**=(Gen, Enc, Dec).
They want to use a symmetric cipher **Sym**, to encipher big plaintexts, by using the key k which was generated by Alice.



1.9 Attacks!!

There are many general types of cryptoanalytic attacks (of course, each of them assumes that the cryptoanalyst has complete knowledge of the encryption algorithm used):

1. **Ciphertext-only attack**. In cryptography, a ciphertext-only attack (COA) or known ciphertext attack is an attack model for cryptanalysis where the attacker is assumed to have access only to a set of ciphertexts. The attack is completely successful if the corresponding plaintexts can be deduced (extracted) or, even better, the key. The ability to obtain any amount of information from the underlying ciphertext is considered a success.

2. **Known-plaintext attack**. The known-plaintext attack (KPA) is an attack model for cryptanalysis where the attacker has samples of both the plaintext and its encrypted version (known as ciphertext version) then they can use them to expose further secret information after calculating the secret key.

3. **Chosen-plaintext attack**. A chosen-plaintext attack (CPA) is a model for cryptanalysis which assumes that the attacker can choose random plaintexts to be encrypted and obtain the corresponding ciphertexts. The goal of the attack is to gain some further information which reduces the security of the encryption scheme. In the worst case, a chosen-plaintext attack could expose secret information after calculating the secret key. Modern cryptography is implemented in software or hardware and is used for a diverse range of applications; for many applications, a chosen-plaintext attack is often very feasible. Chosen-plaintext attacks become extremely important in the context of public key cryptography, where the encryption key is public and attackers can encrypt any plaintext they choose.

4. **Adaptive-chosen-plaintext attack**. A (full) adaptive chosen-ciphertext attack is an attack in which ciphertexts may be chosen adaptively before and after a challenge ciphertext is given to the attacker, with ONE condition that the challenge ciphertext may not itself be queried. This is a stronger attack notion than the lunchtime attack, and is commonly referred to as a CCA2 attack, as compared to a CCA1 (lunchtime) attack.[2] Few practical attacks are of this form. Rather, this model is important for its use in proofs of security against chosen-ciphertext attacks. A proof that attacks in this model are impossible implies that any practical chosen-ciphertext attack cannot be performed.

5. **Chosen-ciphertext attack**. A chosen-ciphertext attack (CCA) is an attack model for cryptanalysis in which the cryptanalyst gathers information, at least in part, by choosing a ciphertext and obtaining its decryption under an unknown key. When a cryptosystem is susceptible to chosen-ciphertext attack, implementers must be careful to avoid situations in which an attackers might be able to decrypt chosen ciphertexts (i.e., avoid providing a decryption scheme).

6. **Chosen-key attack**. Chosen-key attacks are a bit different than other kinds of cryptographic attacks. Usually, they are intended to not just break a cipher but to break the larger system which relies on that cipher.

The attacker should have some knowledge regarding the relationship between various keys that can be used in the cipher. Usually, he knows exactly what keys have been used or he himself can choose the secret key. An example of a chosen-key attack can be a situation when an intruder tries to compromise a hash function based on a block cipher. If the attacker was able to find two different keys which would produce two block cipher outputs that are somehow related to each other, this would mean that the main property of hash functions (never produce predictable output!) had been broken.

7. **Rubber-hose cryptoanalyst**. In cryptography, rubber-hose cryptanalysis is a euphemism for the extraction of cryptographic secrets (e.g. the password to an encrypted file) from a person by coercion or torture such as beating that person with a rubber hose.

1.9.1 Security Level

An encryption algorithm has a security level of n bits if the best known attack requires $O(2n)$ steps. This allows us to compare algorithms and is useful when we combine several primitives in a hybrid cryptosystem to understand any weaknesses. The security level is related to a security parameter λ which is usually written in unary 1^n.
In most cryptographic functions, the key length is an important security parameter.

1.10 CPA-IND experiment

Another type of experiment applicable to symmetric cryptosystems is the CPA-IND (Chosen Plaintext Attack INDistinguishability).

This experiment is similar to the previous one and it is made up different steps:

1. A n-bit key k is generated by running Gen function
2. The adversary (cryptoanalyst), connected to the server (cryptographer) having access to $Enc_k(\cdot)$, chooses two plaintexts P_0 and P_1 of the same length
3. A random bit $b = 0, 1$ is chosen and then a ciphertext C is computed and sent to the adversary.
4. Adversary, who has access to $Enc_k(\cdot)$, generates b'
5. If $b' = b$ then adversary has succeded to distinguish, otherwise not.

The symmetric cryptosystem Π = (Gen, Enc, Dec) is defined as **CPA-IND-secure** (or just CPA-secure) if and only if no adversary $\mathscr{A}$ can distinguish with probability better than $1/2$.

Exercise 1.10.3: The baby ciphers of the previous section are not CPA-secure. For example the Caesar cipher allows full recovery of the secret key, while one-time pad cipher is secure only if key reuse is avoided.

Which are the differences from previous case?
Inside the server there is an $Enc_k(\cdot)$ program running that encrypt according to the symmetric key k and encryption is not deterministic! Before we had that, given a plaintext P, the correspondant ciphertext is always the same. Instead, in this case result could be C now, but then, if we re-use the same encryption function on the same plaintext, it could generate C', C", etc. different from C. This property is due to the fact that encryption is based also on a random sequence of bits:

$$\begin{gathered} \text{C} = Enc_k(random_{bits}||P_b) \\ \text{C'} = Enc_k(random'_{bits}||P_b) \\ \text{C"} = Enc_k(random''_{bits}||P_b) \\ \text{...and so on...} \end{gathered}$$

Note: A good client will knows the length of the random sequence and will throw away the right number of bits in order to keep only the enciphered plaintext. So, for a good client, doesn't matter if it receives C, C', C" or others.

1.11 Appendixes

1.11.1 More about security

In order for cryptography to actually do anything, it has to be embedded in a protocol, written in a programming language, embedded in software, run on an operating system and computer attached to a network, and used by living people. All of those things add vulnerabilities and-more importantly-they're more conventionally balanced.

Security is hard: nothing is really definitively secure.

1.11.2 Security Notions and Goals

Security Notions:

- Perfect Secrecy and One-Time Pad (OTP): In cryptography, the one-time pad (OTP) is an encryption technique that cannot be cracked, but requires the use of a one-time pre-shared key the same size as, or longer than, the message being sent. In this technique, a plaintext is paired with a random secret key (also referred to as a one-time pad). Then, each bit or character of the plaintext is encrypted by combining it with the corresponding bit or character from the pad using modular addition. It has also been proven that any cipher with the property of perfect secrecy must use keys with effectively the same requirements as OTP keys.

- Computational security: A cipher can be said to be computationally secure if it cannot be cracked in 'reasonable time'.
- Provable security: Provable security refers to any type or level of computer security that can be proved. It is used in different ways by different fields. Usually, this refers to mathematical proofs, which are common in cryptography.

Security Goals: There are two main security goals that correspond to different ideas of what it means to learn something about a cipher's behavior:

- Indistinguishability (IND): Ciphertext indistinguishability is a property of many encryption schemes. Intuitively, if a cryptosystem possesses the property of indistinguishability, then an adversary will be unable to distinguish pairs of ciphertexts based on the message they encrypt.
- Non-malleability (NM): Given a ciphertext C_1 (generated starting from the plaintext P_1), it should be impossible to create another ciphertext C_2, whose corresponding plaintext P_2 is related to P_1 in a meaningful way. Surprisingly, the one-time pad is malleable!

1.11.3 Some pre-computer ciphers

Classical ciphers or pre-computers cipher were constructed by using two ideas: Substitution and Permutations.
An example of cipher based upon substitution is the monoalphabetic Caesar cipher.
An example of cipher based upon permutation is the Rail Fence cipher. An example of permutation table is given below:

1	18	11	5	22	28
25	15	8	2	19	12
6	23	29	26	16	9
3	20	13	7	24	30
27	17	10	4	21	14

The table gives a permutation of the set $\{1, 2, \cdots, 30\}$. Here in standard notation:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & \cdots & 30 \\ 1 & 10 & 19 & 28 & 4 & 13 & 22 & 9 & \cdots & 24 \end{bmatrix}$$

Exercise 1.11.7: A knight's tour is a sequence of moves of a knight on a chessboard such that the knight visits every square exactly once. If the knight ends on a square that is one knight's move from the beginning square (so that it could tour the board again immediately, following the same path), the tour is closed; otherwise, it is open. This mathematical problem can be solved by

means of an approach based on an Encryption Algorithm. If we consider an image as a chessboard made up of a lot of cells (pixels) the encryption process is performed in three parts namely Image Padding, Checkerboard Generation and Common-Key XOR-ing (Optional).