Université Libre de Bruxelles
Faculté des Sciences

2022-2023

INFO-F-405: Introduction to cryptography

1. Historical ciphers and general principles

Historical ciphers

Exercise 1

A cipher with $\mathcal{M} = \mathcal{C}$ for certain keys $k \in \mathcal{K}$ is called *involutory* if its encryption and decryption procedures become identical, i.e., for all $m \in \mathcal{M}$, $E_k(m) = D_k(m)$. Under which keys $k \in [0, 25]$ the shift encryption scheme (see slides) becomes involutory?

Answer of exercise 1

For $k = 13$ and $k = 26$. For instance, for $k = 13$ plaintext letter `A` will be encrypted to the ciphertext letter `N`, which is equivalent to computing $14 = 1 + 13 \pmod{26}$ whereas the decryption will compute $1 = 14 + 13 \pmod{26}$. For $k = 13$ the Shift Cipher is sometimes called ROT13. Note that if $k = 26$ the encryption procedure becomes an identity function, i.e. $\#c_i = \#m_i$. That is, using $k = 26$ is not interesting since the ciphertext is identical to the plaintext.

Exercise 2

The following text was encrypted using the shift encryption scheme. Try all possible secret keys to find the one that decrypts the following message:

`Fbzr crbcyr jrne Fhcrezna cnwnznf. Fhcrezna jrnef Puhpx Abeevf cnwnznf.`

How many attempts are needed to be sure to find the correct one?

Assuming the secret key is a uniformly-distributed random variable. What is the probability that the correct key is found after t attempts?

Here is another ciphertext to decrypt:

`'Yvccf, nficu!' - zj fev fw kyv wzijk kyzexj gvfgcv kip kf gizek nyve kyvp cvrie r evn gifxirddzex crexlrxv.`

Answer of exercise 2

The first ciphertext decrypts to `Some people wear Superman pajamas. Superman wears Chuck Norris pajamas.` The secret key is 13.

In this case, exhaustive key search takes up to 26 attempts to find the correct key. If the key is uniformly distributed, the probability to find it after $0 \leq t \leq 26$ attempts is $\frac{t}{26}$.

The second ciphertext decrypts to `'Hello, world!' - is one of the first things people try to print when they learn a new programming language.` The secret key is 17.

Exercise 3

In one of the previous exercices, we performed cryptanalysis by knowing only the ciphertexts. Here we consider known-plaintext attacks where the attacker gets access to some know plaintext/ciphertext pairs and wants to determine the secret key to be able to decrypt past or future messages.

How many pairs of plaintext/ciphertext characters (or bits) must be known in order to determine the secret key without ambiguity in the following ciphers?

1. In the shift encryption scheme?
2. In a general mono-alphabetic substitution scheme?
3. In a general poly-alphabetic substitution scheme (assuming the length t of the key is known)?
4. In the Vigenère cipher (t is known)?
5. In the binary Vigenère cipher (t is known)?

Answer of exercise 3

1. One pair (m_i, c_i) is enough since the same key $k = \#m_i - \#c_i \pmod{26}$ is used to encrypt all characters.
2. To remove any ambiguity, the attacker should collect enough pairs (m_i, c_i) until c_i takes 25 different letter values. This determines the permutation k, as the mapping for the 26-th letter can be deduced from the 25 others.

3. This follows the same idea as for the mono-alphabetic substitution, except that the collection must be done independently for each position i modulo t. E.g., if $t = 2$, the attacker must collect enough pairs (m_i, c_i) until c_i takes 25 different letter values for odd i's, and the same for even i's.

4. Since, in the Vigenère cipher, the key k consists of t characters that are periodically used to encrypt the plaintext, the analyst must known first t characters of the plaintext/ciphertext pair (m, c). Then each of these t characters can be found as in the shift encryption scheme.

5. Similarly, a pair of at least t bits must be known. The key is found by bitwise modulo-2 adding the plaintext and the ciphertext.

Perfect secrecy vs computational security

Exercise 4

A bank defines an encryption algorithm to encrypt their account numbers. An account number is a 12-digit number, i.e., an element of $\mathbb{Z}_{10^{12}}$. Every time an account number is encrypted, a fresh key is chosen randomly and uniformly from the same set $\mathbb{Z}_{10^{12}}$. The encryption goes as follows:

$$E_k(m) = m + k \pmod{10^{12}} \quad D_k(c) = c - k \pmod{10^{12}}$$

Does this cipher satisfy perfect secrecy?

Answer of exercise 4

Yes, as only one key maps a given plaintext m to a given ciphertext c, i.e., $\forall m, c$ there exists exactly one value k s.t. $E_k(m) = c$. So, the probability $\Pr[E_k(m) = c] = \Pr[k \text{ was drawn as key}] = 10^{-12}$ is independent of m.

Exercise 5

What happens if the one-time pad is incorrectly used and that two distinct plaintexts are encrypted with the same key, and why?

1. The key is compromised.

2. The two plaintexts are revealed.

3. The difference between the two plaintexts is revealed.

4. The authenticity of the plaintext is compromised.

Answer of exercise 5

The difference between the two plaintexts is revealed.

In such a case, at each bit position, when the bits of the two plaintext are equal (resp. different), the ciphertext bits are equal (resp. different). Without knowing their individual values, the adversary can nevertheless tell where the two messages are equal and where they are different.

Mathematically, bitwise adding modulo-2 the two ciphertexts cancels the contribution of the key and reveals the bitwise difference between the two plaintexts (see the slides).

Exercise 6

Suppose that the probability of wining the lottery is $\frac{1}{10^6}$. What is more likely guessing an AES-128 key on the first try or winning the lottery 6 times in a row? What about winning the lottery 7 times in a row?

Answer of exercise 6

Guessing an AES-128 key:

$$\frac{1}{2^{128}} = \frac{1}{2^8 \times 2^{120}} = \frac{1}{2^8 \times (2^{10})^{12}} = \frac{1}{256 \times (1024)^{12}}$$

Winning n times in a row:

$$\left(\frac{1}{10^6}\right)^n = \frac{1}{(10^6)^n} = \frac{1}{((10^3)^2)^n} = \frac{1}{(1000)^{2n}}$$

Exercise 7

Assume an adversary performs an exhaustive key search on a huge network of 10^9 computers, each capable of testing 10^9 keys per second. After about how much time will a 128-bit key typically be found?

1. A few seconds.
2. A few days.
3. A few years.

4. A few centuries.

5. A few times the age of the universe.

Answer of exercise 7

A few times the age of the universe.

There are 2^{128} keys to test, and on average the key will be found after scanning half the key space, so after testing $2^{128}/2 = 2^{127}$ keys. With this network, the adversary can test 10^{18} keys per second. So it will take $2^{127}/10^{18}$ seconds. Divide by 3600 to convert to hours, by 24 to days, by 365 to years, etc.

Security definitions

Exercise 8

Suppose that (Gen, E, D) is an IND-CPA secure probabilistic encryption scheme, i.e., there is no known way of winning the IND-CPA game other than with a probability negligibly close to $\frac{1}{2}$ or with over-astronomical resources. Let the key and plaintext spaces be $\{0,1\}^n$ with $n \geq 128$. Point out the IND-CPA secure schemes in the following list:

- $E'_k(m) = E_k(m)||\text{first bit}(m)$
- $E'_k(m) = (\text{last bit}(m) \oplus \text{first bit}(m))||E_k(m)$
- $E'_k(m) = E_k(m)||1$
- $E'_k(m) = E_k(m)||E_k(m)$
  (+a side-question: will the first and last n bits of $E'_k(m)$ always be equal?)
- $E'_k(m) = k||E_k(m)$
- $E'_k(m) = E_k(m)||(k \oplus 1^n)$
- $E'_{k1,k2}(m) = E_{k1}(m)||E_{k2}(m)$
- $E'_k(m) = E_{0^n}(m)$
- $E'_k(m) = E_k(m)||(m \oplus k)$

For each scheme that is *not* IND-CPA secure, please specify how an adversary can win the IND-CPA game, i.e., what the adversary chooses as plaintexts m_0 and m_1, what are the queries to be made before or after this choice, how the adversary distinguishes between the ciphertexts of m_0 and m_1.

Answer of exercise 8

- $E'_k(m) = E_k(m)||\text{first bit}(m)$: The adversary chooses m_0 and m_1 such that they differ in the first bit. It can recognize which plaintext was encrypted thanks to first bit(m) exposed in the ciphertext. No need for queries during the game.

- $E'_k(m) = (\text{last bit}(m) \oplus \text{first bit}(m))||E_k(m)$: Same reasoning, except that the adversary chooses m_0 and m_1 such that they differ in either the first or the last bit (but not both).

+ $E'_k(m) = E_k(m)||1$: The extra bit 1 present in all ciphertexts do not help the adversary, so E' is IND-CPA secure.

+ $E'_k(m) = E_k(m)||E_k(m)$: The two ciphertexts do not help the adversary, so E' is IND-CPA secure. (The answer to the side-question is no: Not only the scheme is said to be probabilistic, if (E, D) is IND-CPA secure, the adversary cannot even tell whether the same plaintext was encrypted twice. This means that the ciphertext space can be larger than the plaintext space, since one plaintext can be mapped to several ciphertexts.)

- $E'_k(m) = k||E_k(m)$: The key is revealed in the ciphertext, removing any security. To win the game, it suffices the adversary to use the revealed key to decrypt $E_k(m_0$ or $m_1)$ and find out which one it was.

- $E'_k(m) = E_k(m)||(k \oplus 1^n)$: Same reasoning, the key is also revealed since the mask 1^n is known.

+ $E'_{k1,k2}(m) = E_{k1}(m)||E_{k2}(m)$: The notation here suggests that the new encryption scheme has a $2n$-bit key, but each n-bit half is used independently. The adversary has twice more chance of winning the IND-CPA on E' than on E, since distinguishing either $E_{k1}(m_0$ or $m_1)$ or $E_{k2}(m_0$ or $m_1)$ is sufficient. Twice a neglible probability is still a neglible probability, hence E' is IND-CPA secure, although 1-bit less secure than E.

- $E'_k(m) = E_{0^n}(m)$: The scheme E' does not make use of its secret key. Anything encrypted with E_{0^n} can therefore be decrypted by the adversary with D_{0^n}, and it can immediately distinguish $E_{0^n}(m_0$ or $m_1)$ to win the game.

- $E'_k(m) = E_k(m)||(m \oplus k)$: As the plaintexts are chosen by the adversary, $m \oplus k$ reveals the key.

Exercise 9

Let (Gen, E, D) be a symmetric-key encryption scheme with diversification that is IND-CPA-secure. The definition of IND-CPA requires that the adversary respects the condition that the diversifier d is a nonce. What would happen if this condition was not respected? Show how he can win this variant of IND-CPA game where d is not unique, with the least number of queries.

Answer of exercise 9

The adversary chooses one plaintext m_0 and one diversifier value d arbitrarily, and he asks for $c_0 = \text{Enc}_k(d, m_0)$. The adversary chooses another plaintext $m_1 \neq m_0$ but with the same length $|m_0| = |m_1|$, and submits d, m_0 and m_1 to the challenger. The challenger sends back $c = \text{Enc}_k(d, m_b)$ with unknown b. If $c = c_0$, then it means m_0 was encrypted and $b = 0$, otherwise it must be m_1 and $b = 1$.

The trick here is that the adversary uses the same value d for his first query and for the encryption of the challenger. In the actual IND-CPA game, this is not allowed, and the encryption of m_b by the challenger would be done with a different diversifier.

Exercise 10

Suppose that (Gen, E, D) is an IND-CPA secure symmetric-key encryption scheme with diversification, i.e., there is no known way of winning the IND-CPA game other than with a probability negligibly close to $\frac{1}{2}$ or with over-astronomical resources. Let the key space be $\{0,1\}^n$ with $n \geq 128$, the plaintext space to be arbitrary and the diversifier space be the set of non-negative integers $\mathbb{N}$. Point out the IND-CPA secure schemes in the following list:

- $E'_k(d, m) = E_k(0, m)$
- $E'_k(d, m) = E_k(d + 1, m)$
- $E'_k(d, m) = E_k(d, m)\|0^d$
- $E'_k(d, m) = E_k(\lfloor d/2 \rfloor, m)$
- $E'_k(d, m) = E_{0^d\|1^{n-d}}(d, m)$
- $E'_k(d, m) = E_k(\text{random} \in \{0, 1, \ldots, 36\}, m)$

- $E'_k(d, m) = E_k(\text{random} \in \{0, 1, \ldots, 2^{256}\}, m)$

For each scheme that is *not* IND-CPA secure, please specify how an adversary can win the IND-CPA game, i.e., what the adversary chooses as diversifiers, as plaintexts m_0 and m_1, what are the queries to be made before or after this choice, how the adversary distinguishes between the ciphertexts of m_0 and m_1.

Answer of exercise 10

- \- $E'_k(d, m) = E_k(0, m)$: Even if the adversary uses unique values for d in E', the diversifier is ignored and repeated internally when calling E. Hence, the encryption of identical plaintexts can be recognized. The adversary can submit for m_0 a previously queries plaintext and for m_1 an arbitrary one.

- \+ $E'_k(d, m) = E_k(d + 1, m)$: The diversifier is just invertibly remapped to a different value when calling E, so E' is as secure as E.

- \+ $E'_k(d, m) = E_k(d, m) \| 0^d$: The value d is public, so disclosing it in the ciphertext does not help the adversary.

- \- $E'_k(d, m) = E_k(\lfloor d/2 \rfloor, m)$: Both $d = 2n$ and $d = 2n + 1$ will be mapped to n when using E. Hence, even if the adversary respects the unicity of d in E', the encryption of identical plaintexts can be recognized for pairs of queries. Concretely, the adversary can ask for one query with $d = 0$ and m_0, then submit $d = 1$, m_0 and an arbitrary m_1 to the challenger.

- \- $E'_k(d, m) = E_{0^d \| 1^{n-d}}(d, m)$: Here, E is called with a publicly known key and the adversary can decrypt the ciphertext with $D_{0^d \| 1^{n-d}}$ to win the game.

- \- $E'_k(d, m) = E_k(\text{random} \in \{0, 1, \ldots, 36\}, m)$: In this case, E is called with a random value instead of d. Given the small set of possible values, the adversary can expect that the diversifier used in E will repeat. Concretely, the adversary can ask for the encryption of m_0 a hundred times or so, and with a high probability he will get all the possible ciphertexts corresponding to m_0. He then submits m_0 and an arbitrary m_1 and d. If c is one of the possible ciphertexts for m_0, $b = 0$; otherwise, $b = 1$.

- \+ $E'_k(d, m) = E_k(\text{random} \in \{0, 1, \ldots, 2^{256}\}, m)$. This case is similar, but the cardinality of the random values used in E is huge. Assuming a uniform distribution, the repetition of the random value is an event with a negligible probability and the adversary has only a negligible advantage over a random guess.

Exercise 11

A bank issues a smartcard to each of its N customers. Each smartcard i contains its own secret key K_i of k bits. Because of the way the protocol was designed, each smartcard encrypts the string `"hello"` with its secret key and sends it over the network.

An attacker collects all these N ciphertexts and starts a *multi-target exhaustive key search*. In more details, the attacker encrypts the string `"hello"` with each possible key in the key space until he gets a ciphertext that matches one of the collected ones. When it does, it probably means that he hits the secret key of one of the smartcards.

1. What is the security strength of the scheme under this attack as a function of k and N?
2. If a bank issues a billion ($N \approx 2^{30}$) smartcards, each with its own $k = 128$-bit key, is the scheme still secure in practice?
3. What could the bank do to have a security strength of 128 bits?

Answer of exercise 11

1. Each attempt has a chance $\frac{N}{2^k}$ to hit one of the N keys. After t attempts, the probabilty is $\epsilon(t) \approx \frac{tN}{2^k}$. We have

$$\log_2 t - \log_2 \epsilon(t) = \log_2 t - \log_2 \frac{tN}{2^k} = k - \log_2 N \geq s,$$

   hence the security strength is $k - \log_2 N$ in the absence of more efficient attacks. There is therefore a reduction of $\log_2 N$ bits of security compared to the nominal exhaustive key search.

2. If $N = 2^{30}$ and $k = 128$, the security strength is $128 - 30 = 98$ bits. It would take about 2^{98} attempts to hit one of the keys, which is still infeasible in practice today.

3. The bank could either increase the key length to 158 bits to compensate for the 30-bit loss, or fix its protocol so as to prevent multi-target exhaustive key search. For this latter case, the protocol could make the encryption depend on the smartcard identifier i in some way (e.g., by including i as associated data with authenticated encryption, see the appropriate chapter for more details). This forces the attacker to choose a value i, and therefore test only that K_i, at each attempt.

Exercise 12

An automated teller machine (ATM) distributes cash to the customers of a bank. It is connected to the bank's server, which is in charge of authorizing and tracking transactions. The bank and the ATM share a k-bit secret key K. When authorizing, the bank sends the ATM a triplet (u, m, tag) allowing user u to receive an amount m in cash, and tag is a n-bit message authentication code (MAC) computed under key K over the first two components of the triplet. (This is of course a very simplified view for the sake of this exercise.)

1. What is the bank trying to protect against, assuming an adversary who has full access to the network connecting the bank and the ATM?

2. Let us assume for now that the bank and the ATM use a secure MAC function. What is the minimum key length k that is required to have a security strength of 128 bits, and why?

3. Let us further assume that the ATM processes not more than one triplet per second and that the secret key K is securely renewed every 24 hours. What is the minimum length n of the MAC such that the probability of fraudulent transaction is less than 10^{-12} per day, and why?

Answer of exercise 12

1. Without authentication, an attacker who has access to the network could insert messages that authorize fraudulent transactions. The MAC allows the ATM to distinguish fraudulent triplets from those legitimately created by the bank.

2. The minimum key size is 128 bits. With a smaller key size, exhaustive key search could be done with a time t and a success probability ϵ that would violate the equation $\log_2 t - \log_2 \epsilon \geq s = 128$.

3. This is about random tag guessing. For a given key, the attacker can try to guess a tag $24 \times 3600 = 86400$ times; after that, the key is renewed and the attacker must start afresh. Correctly guessing a tag of n bits succeeds with a probability 2^{-n} per attempt. To keep the probability of a correct key guess below 10^{-12} per day, we need to satisfy:

$$\begin{aligned} 86400 \times 2^{-n} &\leq 10^{-12} \\ 2^{-n} &\leq 10^{-12}/86400 \\ -n &\leq \log_2(10^{-12}/86400) \\ n &\geq \log_2(10^{12} \times 86400) \approx 56.3 \end{aligned}$$

Hence, the minimum tag size is 57 bits.

Others

Exercise 13

In order to save space or bandwidth we often use compression algorithms. Suppose you want to use data compression with encryption. How should these two operations be combined? Why?

Answer of exercise 13

Compress then encrypt. An encrypted message looks like random data and the compression algorithm will not be able to reduce the size.

Exercise 14

Imagine that you have to use a relatively unreliable network i.e., packets often arrive with errors (bit flips) to their destination. Suppose you want to use an error correction code with encryption. How should these two operations be combined? Why?

Answer of exercise 14

Encrypt then apply error correction code. If the ciphertext is corrupted, the decryption algorithm will not work and there is no way to retrieve any meaningful information.

Exercise 15

It is possible to use a symmetric crypto system to encrypt a short message e.g., 40-bit and obtain a 40-bit ciphertext. Could we create a secure public key crypto system with short ciphertexts?

Answer of exercise 15

No. Public key is known and one could encrypt all possible messages of short size.

Exercise 16

A company that installs and maintains an open-source operating system is creating an automatic update mechanism for its customers. Each computer connected to the internet can fetch an *update pack* containing the latest changes to be made to the operating system. The company would like to guarantee the integrity of these

update packs so as to avoid the installation of malicious software on their customers' computers.

Please propose a solution based on schemes such as encryption, either public-key or secret-key and authentication, either MAC or signature. You also need to describe how the keys are distributed. Would a secret-key or a public-key approach be more appropriate?

Answer of exercise 16

As the operating system is open source, there is no confidentiality required, i.e., the update packs are distributed in the clear. However, their authenticity must be guaranteed, otherwise an adversary could install malicious software. Therefore, the update packs should come with a tag for the customers to be able to verify its authenticity.

If we go for a secret-key approach, this means that the authenticity is ensured with a message authentication code (MAC) computed over the update pack with a key secretely shared between a particular customer and the company. This means that the company must establish as many secret keys as it has customers. Perhaps the company can choose a secret key per customer and send it securely via a secure channel, but it is not so clear how.

If we go for a public-key approach, this means that the authenticity is sensured with a signature computed over the update pack with the company's private key, which can be verified with the company's public key. Here, a single private/public key pair is necessary. The company's public key can be bundled together with the first version of the operating system. Of course, the customer must be sure that the public key really belongs to the company, but this problem is not harder than ensuring that the first version of the operating system is genuine.

For the given reasons, the public key approach seems easier to manage: only one key instead of one per customer, and the distribution of the public key does not need confidentiality. Yet, the public-key approach still relies on the fact that the first version of the operating system and the company's public key arrive without corruption on the customer's computer.

Exercise 17

Suppose Alice is broadcasting packets to 6 recipients $B_1, , B_6$. Privacy is not important but integrity is. In other words, each of $B_1, , B_6$ should be assured that the packets he is receiving were sent by Alice.

Alice decides to use a MAC. Suppose Alice and $B_1, , B_6$ all share a secret key k. Alice computes a tag for every packet she sends using key k. Each user B_i verifies the tag

when receiving the packet and drops the packet if the tag is invalid. Alice notices that this scheme is insecure because user B_1 can use the key k to send packets with a valid tag to users $B_2, , B_6$ and they will all be fooled into thinking that these packets are from Alice.

Instead, Alice sets up a set of 4 secret keys $S = \{k_1, , k_4\}$. She gives each user B_i some subset $S_i \subseteq S$ of the keys. When Alice transmits a packet she appends 4 tags to it by computing the tag with each of her 4 keys. When user Bi receives a packet he accepts it as valid only if all tags corresponding to his keys in S_i are valid. For example, if user B_1 is given keys $\{k_1, k_2\}$ he will accept an incoming packet only if the first and second tags are valid. Note that B_1 cannot validate the 3rd and 4th tags because he does not have k_3 or k_4.

How should Alice assign keys to the 6 users so that no single user can forge packets on behalf of Alice and fool some other user? How many keys are needed for t users if each of them receive only two keys?

Answer of exercise 17

$\{k_1, k_2\}, \{k_2, k_3\}, \{k_3, k_4\}, \{k_1, k_3\}, \{k_2, k_4\}, \{k_1, k_4\}$
Number of keys : $\lceil (1 + \sqrt{1 + 8t})/2 \rceil$

Université Libre de Bruxelles
Faculté des Sciences

2023-2024

INFO-F-405: Introduction to cryptography

Introduction to the finite field GF(2^8)

The goal of this session is to recall the basic properties of the field GF(256) used in AES .

Finite fields are algebraic structures in which one can apply the basic operations of addition, subtraction, multiplication and division with their usual properties (e.g. distribution). One example of finite fields is GF(5) (also noted $\mathbb{Z}/5\mathbb{Z}$ or $\mathbb{F}_5$), the field of integers modulo 5. Indeed, in this field, one can:

- Add two elements: $4 + 3 \equiv 2 \mod 5$
- Subtract two elements: $2 - 3 \equiv 4 \mod 5$
- Multiply two elements: $3 \cdot 2 \equiv 1 \mod 5$
- Divide one element by another (where division is seen as multiplication by an inverse): $4/2 = 4 \cdot 2^{-1} = 4 \cdot 3 \equiv 2 \mod 5$

For any $n \in \mathbb{Z}$, the three first operations do exists in $\mathbb{Z}/n\mathbb{Z}$ and those structures are called rings. Division is fully defined when all nonzero elements have an inverse, which is only the case when n is prime. Actually, all the finite fields have p^i elements for a prime p and the particular case of prime order fields (when $i = 0$) are (isomorphic to) the $\mathbb{Z}/p\mathbb{Z}$ and are written GF(p) or $\mathbb{F}_p$. To construct finite fields of nonprime orders p^i for $i \neq 0$, one has to work with polynomials over GF(p) with operations modulo an irreducible polynomials of degree i.
In the following, we will construct the field GF(256=2^8).

1 The polynomial ring GF(2)[X]

We start with GF(2) which is simply the set $\{0, 1\}$ with addition and multiplication modulo 2. Those two operations can alternatively seen as the logical XOR and AND operations.

Addition in $\mathrm{GF}(2)$	**Multiplication in** $\mathrm{GF}(2)$
$0+0=0$	$0\cdot 0=0$
$0+1=1$	$0\cdot 1=0$
$1+0=1$	$1\cdot 0=0$
$1+1=0$	$1\cdot 1=1$

We then construct $\mathrm{GF}(2)[X]$ which is the ring of polynomials with coefficients in $\mathrm{GF}(2)$. An example of an element of this ring is $X^{15}+X^3+X^1+X^0 \in \mathrm{GF}(2)[X]$.

Encoding

Such polynomials can be represented as a binary string by concatenating all coefficients (which are in $\{0,1\}$). This binary string can subsequently be encoded in hexadecimal for a compact representation. For example $X^6+X^4+X+1 = 1X^6+0X^5+1X^4+0X^3+0X^2+1X+1$ can be written `1010011` in binary and `0x53` in hexadecimal.

Addition/subtraction

Addition and subtraction are performed coefficient-wise on the polynomials in the field $\mathrm{GF}(2)$.
For example, X^3+X^2+1 added to X^2+X+1 is:

$$(1X^3+1X^2+0X+1)+(0X^3+1X^2+1X+1) = (1+0)X^3+(1+1)X^2+(0+1)X+(1+1) = X^3+X.$$

Alternatively, this can be written `0xD + 0x7 = 0xA` $\in \mathrm{GF}(2)[X]$

Multiplication

Multiplication is the usual polynomial multiplication except that coefficients are reduced modulo 2. For example, (X^2+X) multiplied by (X^6+1) is:

$$(X^2\cdot X^6)+(X\cdot X^6)+(X^2\cdot 1)+(X\cdot 1) = X^8+X^7+X^2+X.$$

Alternatively, this can be written `0x6 · 0x41 = 0x186` $\in \mathrm{GF}(2)[X]$

Modular reduction of polynomials

For a given polynomial $m(X)$. A polynomial $a(X)$ can always be written:

$$a(X) = b(X)m(X) + r(X),$$

where the degree of $r(X)$ is less than the degree of $m(X)$. Then, we can say that $a(X)$ is congruent to $r(X)$ modulo $m(X)$, and write it

$$r(X) \equiv a(X) \mod m(X).$$

To reduce $a(X)$ modulo $m(X)$, the simplest procedure is to repeatedly add or subtract a multiple of $m(X)$ to make the highest degree term disappear until obtaining a polynomial of degree less than the degree of $m(X)$.

Exercices

Exercice 1. Write the polynomial corresponding to `0x7A` in hexadecimal.

Solution. $X^6 + X^5 + X^4 + X^3 + X$

Exercice 2. What is the hexadecimal representation of $X^7 + X^6 + X^2 + 1$?

Solution. `0xC5`

Exercice 3. Add $X^4 + X^3 + 1$ with $X^7 + X^4 + X^2 + 1$ in $\mathrm{GF}(2)[X]$. Rewrite the addition in hexadecimal representation. Which well-known operation has actually been performed between the hexadecimal values?

Solution. $X^7 + X^3 + X^2$
`0x19` + `0x95` = `0x8C` (XOR)

Exercice 4. Multiply $X^3 + X^2$ with $X^{17} + X^{16} + X + 1$.

Solution. $X^{20} + X^{18} + X^4 + X^2$

Exercice 5. Let $m(X) = X^8 + X^4 + X^3 + X + 1$. Reduce the following polynomials modulo $m(X)$:

1. $X^8 + X^7 + X^3 + 1$
2. $X^{10} + X^9 + X^8$
3. $X^{108} + X^{104} + X^{103} + X^{101} + X^{100}$

Solution.

1. $X^7 + X^4 + X$
2. $X^6 + 1$
3. 0

2 $\mathrm{GF}(2^8)$: finite field of order 256

There exists multiple ways to construct $\mathrm{GF}(2^8)$. The one we are using in this document was not randomly chosen, we follow the construction used to define the algorithm Rijndael (AES).

Here, the field $\mathrm{GF}(2^8)$ is defined as the set of binary polynomials of degree less than 8. Addition is identical to the one in $\mathrm{GF}(2)[X]$ because the result of the addition of two polynomials of degree less than 8 is also of degree less than 8. Unlike addition, the multiplication between two degree less than 8 polynomials results in polynomials of degrees up to 14. Hence, we shall reduce modulo a degree 8 polynomials. In particular, we will reduce modulo an irreducible polynomials to get the field structure.

2.1 Multiplication

Let $m(X) = X^8 + X^4 + X^3 + X + 1$ be the irreducible polynomial used in Rijndael. In this representation of $\mathrm{GF}(2^8)$, multiplying two polynomials $a(X)$ and $b(X)$ is done by:

- First multiplying in $\mathrm{GF}(2)[X]$
- Then reduce the result modulo $m(X)$.

2.2 Multiplicative inverse

As explained in the intro, the term *field* indicates that we are working with an object with specific algebraic properties. In particular, addition and multiplication should be invertible. For example, the reals numbers with addition and multiplication form a field. Every element has an additive inverse ($a + (-a) = 0, \forall a \in \mathbb{R}$), and every nonzero element has a multiplicative inverse ($a \cdot \frac{1}{a} = a \cdot a^{-1} = 1, \forall a \in \mathbb{R} \setminus \{0\}$).

Inverting addition in GF(2^8) is trivial because elements are self inverse: $a + a = 0$, for all a.

For multiplication, every element except 0 has an inverse. For any polynomial $a \in$ GF(2^8), $a \neq 0$, there always exist another polynomial a^{-1} such that $a \times a^{-1} = a^{-1} \times a = 1$.

2.3 Exercices

Exercice 7. Multiplication by X. Perform the following multiplication in GF(2^8):

1. $X \cdot X^3$
2. $X \cdot (X^6 + X^2 + 1)$
3. $X \cdot X^7$, then $X \cdot (X \cdot X^7)$
4. $X \cdot (X^7 + X^3 + X^2 + 1)$

Solution.

1. X^4
2. $X^7 + X^3 + 1$
3. $X \cdot X^7 \equiv X^4 + X^3 + X + 1$ and $X \cdot X \cdot X^7 \equiv X^5 + X^4 + X^2 + X$
4. 1

Exercice 8. Using results of the last exercise, compute $(X^2 + X + 1)X^7$.

Solution. $X^7 + X^5 + X^3 + X^2 + 1$

Exercice 9. Multiply `0x8A` by `0xD5`.

Solution. $X^7 + X^6 + X^5 + X^4 + X^3 + X$

Exercice 10. Compute X^{-1}, that is the say the inverse of X. Explain how to compute the inverse of X^2.

Solution. $X^{-1} \equiv X^7 + X^3 + X^2 + 1$
$X^{-2} \equiv X^7 + X^6 + X^3 + X + 1$

2.4 Extra exercice (harder)

The following exercice has been exctracted from the following book: Baigneres, Th., Junod, P., Lu, Y., Monnerat, J., Vaudenay, S., "A Classical Introduction to Cryptography Exercise Book" (2006)

In AES, the main diffusion step is a linear application defined as follows. The 32-bit blocks are considered as polynomials of degree smaller than 4 over GF(256). This linea application consists in multiplying the input polynomial $A(Y) = a_3 \cdot Y^3 + a_2 \cdot Y^2 + a_1 \cdot Y + a_0$ with the fixed polynomial $C \in$GF(256)$[Y]$ = 0x03 $\cdot Y^3 +$ 0x01 $\cdot Y^2 +$ 0x01 $\cdot Y +$ 0x02 modulo the polynomial $Y^4 + 1$ defined in GF(256)$[Y]$ as well.
Compute the image of 0x836F13DD (where $a_0 =$ 0x83, $a_1 =$ 0x6F, $a_2 =$ 0x13, $a_3 =$ 0xDD under this diffusion step.
(Hint: computing modulo $Y^4 + 1$ means $Y^4 + 1 \equiv 0$ and $1 \equiv -1 \mod 2$)

Université Libre de Bruxelles
Faculté des Sciences

2022-2023

INFO-F-405: Introduction to cryptography

2. Symmetric-key techniques

Keystream generators and stream ciphers

Exercise 1

What happens if the diversifier (or nonce) is not unique, i.e., if the stream cipher is incorrectly used and that two distinct plaintexts are encrypted with the same key and the same diversifier? What consequences does it have in the case of a known plaintext attack?

Answer of exercise 1

Like for a misused one-time pad, the difference between the two plaintexts is revealed.

In the case of a known plaintext attack, one of the two plaintexts can be known, and if the difference is known, the other one is revealed.

Exercise 2

Let us consider two linear feedback shift registers (LFSRs) of 4 bits. For each, simulate its execution and determine the cycles. What is the period of the longest possible cycle? Which LFSR has it?

1. $1 + D^2 + D^4$: an iteration is $(b_1, b_2, b_3, b_4) \leftarrow (b_2 + b_4, b_1, b_2, b_3)$.
2. $1 + D^3 + D^4$: an iteration is $(b_1, b_2, b_3, b_4) \leftarrow (b_3 + b_4, b_1, b_2, b_3)$.

Answer of exercise 2

For $1+D^2+D^4$, there are two cycles of period 6, one of period 3 and one of period 1.

- Period of 6: `1000` $\rightarrow$ `0100` $\rightarrow$ `1010` $\rightarrow$ `0101` $\rightarrow$ `0010` $\rightarrow$ `0001` and back.

- Period of 6: `1100` $\rightarrow$ `1110` $\rightarrow$ `1111` $\rightarrow$ `0111` $\rightarrow$ `0011` $\rightarrow$ `1001` and back.
- Period of 3: `0110` $\rightarrow$ `1011` $\rightarrow$ `1101` and back.
- And of course `0000` stays on itself.

The longest possible cycle has period $2^4 - 1 = 15$, and $1 + D^3 + D^4$ has it: `1000` $\rightarrow$ `0100` $\rightarrow$ `0010` $\rightarrow$ `1001` $\rightarrow$ `1100` $\rightarrow$ `0110` $\rightarrow$ `1011` $\rightarrow$ `0101` $\rightarrow$ `1010` $\rightarrow$ `1101` $\rightarrow$ `1110` $\rightarrow$ `1111` $\rightarrow$ `0111` $\rightarrow$ `0011` $\rightarrow$ `0001` and back, plus of course `0000` that stays on itself.

Exercise 3

The Mantin-Shamir attack on RC4 shows that the second byte of keystream has a probability of about $\frac{2}{257}$ of taking value 0 and a probability of about $\frac{1}{257}$ of taking each of the other 255 possible values. What is the probability of winning the IND-CPA game when exploiting this property? Give an upper bound on the security strength s of RC4.

Answer of exercise 3

Since the second byte of keystream is biased towards value 0, the second byte of ciphertext has slightly more chance of being equal to the second byte of the plaintext than to another value.

We therefore assume the following strategy for the adversary. The adversary chooses the plaintext m_0 (resp. m_1) such that the second byte has value a (resp. b), with $a \neq b$. When getting the second byte c of the encryption of m_0 or m_1, the adversary says that the plaintext is m_0 (resp. m_1) when $c = a$ (resp. $c = b$), and guesses randomly when $c \notin \{a, b\}$.

$$\begin{aligned}\Pr[\text{win}] = \Pr[\text{challenger encrypts } m_0](&\Pr[c = a \mid \text{challenger encrypts } m_0] \\ &+\frac{1}{2}\Pr[c \notin \{a,b\} \mid \text{challenger encrypts } m_0]) \\ +\Pr[\text{challenger encrypts } m_1](&\Pr[c = b \mid \text{challenger encrypts } m_1] \\ &+\frac{1}{2}\Pr[c \notin \{a,b\} \mid \text{challenger encrypts } m_1]).\end{aligned}$$

The challenger will choose m_0 or m_1 each with probability $\frac{1}{2}$. Therefore, by symmetry, we have

$$\begin{aligned}\Pr[\text{win}] &= \Pr[c = a \mid \text{challenger encrypts } m_0] + \frac{1}{2}\Pr[c \notin \{a,b\} \mid \text{challenger encrypts } m_0] \\ &\approx \frac{2}{257} + \frac{1}{2} \times \frac{254}{257} \\ &= \frac{258}{514} = \frac{1}{2} + \frac{1}{514}.\end{aligned}$$

The advantage is therefore $\epsilon = \frac{1}{514}$ and, considering the negligible amount of computations needed to mount this attack, we set $\log_2(t+d) = 0$ and deduce that RC4 has at most (a little over) 9 bits of IND-CPA security.

Note that the attack does not require to be in the *chosen plaintext* model, a *ciphertext-only* model is sufficient.

Block ciphers

Exercise 4

Complementating a bit string means changing the value of each bit or, equivalently, adding 1 to all bits (modulo 2 of course). Mathematically, if x is a bit string of length n, the complementation of x can be written as :

$$\overline{x} = x \oplus 1^n.$$

Let x and y be two bit strings of length n. Can you simplify the following expressions?

- $\overline{x} + \overline{y}$
- $\overline{x + y}$
- $\overline{(x \| y)}$
- $f(\overline{x})$,

where f is a bit transposition from n bits to n bits. Note that in this case we have $f(1^n) = 1^n$.

Answer of exercise 4

Using the definition :

- $\overline{x} \oplus \overline{y} = x \oplus 1^n \oplus y \oplus 1^n$
  $= x \oplus y \oplus (1^n \oplus 1^n)$
  $= x \oplus y$

- $\overline{x \oplus y} = x \oplus y \oplus 1^n$
  $= \overline{x} \oplus y \quad \text{or} \quad x \oplus \overline{y}$

- $\overline{(x\|y)} = x\|y \oplus 1^{2n}$ (as $x\|y$ is $2n$-bit long)
  $= x\|y \oplus 1^n\|1^n$
  $= (x \oplus 1^n)\|(y \oplus 1^n)$
  $= \overline{x}\|\overline{y}$

- $f(\overline{x}) = f(x \oplus 1^n)$
  $= f(x) \oplus f(1^n)$ (since f is linear)
  $= f(x) \oplus 1^n$ (since f is a bit transposition)
  $= \overline{f(x)}$

Exercise 5

Complementating a bit string means changing the value of each bit, or equivalently, adding 1 to all bits (modulo 2 of course). DES has a non-ideal property: If the input block and the key are complemented, then the output block is complemented as well. In other words, if $\text{DES}_k(x) = y$, then $\text{DES}_{\overline{k}}(\overline{x}) = \overline{y}$, with the overline to indicate complementation.

The goal of this exercise is to describe what happens inside the DES when input block and the key are complemented. (And by "what happens", it is meant "how do the processed values change when computing $\text{DES}_{\overline{k}}(\overline{x})$ instead of $\text{DES}_k(x) = y$".)

1. If the key is complemented, what happens to the subkeys?
2. If the input block is complemented, what happens to the left and right parts at the beginning of the Feistel network?
3. Consider the first round. The right part R enters the f function and goes through the expansion E. What happens to the output of E?
4. Still inside the f function, what happens when $E(R)$ is XORed with the subkey K?
5. What happens at to the left part L after being XORed with $f(R)$?
6. What happens in the remaining rounds and to the output?

7. Does this complementation property involve a particular property of the S-boxes?

Answer of exercise 5

1. If the key is complemented, all the subkeys are complemented. This is because the subkeys are obtained by a bit-transposition of the key, so this preserves the complementation.
2. When the input block is complemented, the left and right halves L and R are also complemented.
3. When R enters the f function, it is complemented, and it remains complemented after the expansion E.
4. When $E(R)$ and the subkey K are XORed together, both complementations cancel each other. To see this, let's look at the bit level, and say we compute $\overline{x} + \overline{y}$. Then, $\overline{x} + \overline{y} = (x + 1) + (y + 1) = x + y$ in $\mathbb{Z}_2$. So, the input of the S-boxes is *not* complemented anymore and therefore the output of the f function has the same value as before R and K were complemented.
5. The output of the f function is then XORed to the left half L, which is complented and remains so, since $\overline{x} + y = x + 1 + y = \overline{x + y}$.
6. The output of the round is therefore complemented, and the same reasoning can be extended to the 16 rounds of DES.
7. This property does not involve a particular property of the S-box, as the input of the S-box is unchanged after the complementation.

Exercise 6

During each round of AES, the state (consisting of a 4×4 matrix of elements of GF(256), represented as bytes) undergoes the function MixColumns, which is meant to produce diffusion among the bytes of the state. For each column of the state, this function consists in a simple multiplication between the column and a fixed matrix $\mathbf{M}$. If $(s_0, s_1, s_2, s_3)^\top$ is the input of the function (i.e., a column of the state), we get the output $(b_0, b_1, b_2, b_3)^\top$:

$$\begin{pmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{02} & \texttt{03} & \texttt{01} \\ \texttt{01} & \texttt{01} & \texttt{02} & \texttt{03} \\ \texttt{03} & \texttt{01} & \texttt{01} & \texttt{02} \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

What is the output of MixColumns for the column input $s = (\texttt{37}, \texttt{21}, \texttt{A5}, \texttt{C0})^\top$?

Answer of exercise 6

We can compute every byte b_i one by one. For b_0, the matrix multiplication yields

$$b_0 = (\texttt{02} \times \texttt{37}) + (\texttt{03} \times \texttt{21}) + (\texttt{01} \times \texttt{A5}) + (\texttt{01} \times \texttt{C0})$$

Recall that the hexadecimal notation is a compact way to express polynomials in GF(256). In this notation, an addition over GF(256) corresponds to a bitwise XOR over the bytes.

Before we do the multiplications, we have to translate the bytes into polynomials. The multiplication of the polynomials is done modulo $X^8 + X^4 + X^3 + X + 1$. If we wish to write the result in the hexadecimal notation, we have to translate back the resulting polynomial into the hexadecimal notation.

- $\texttt{02} \times \texttt{37} = 0000\,0010 \times 0011\,0111$
  $= X \times (X^5 + X^4 + X^2 + X + 1)$
  $= X^6 + X^5 + X^3 + X^2 + X$
  $= 0110\,1110 = \texttt{6E}$

Note that multiplying by $\texttt{02}$ does nothing but shifting the bits of $\texttt{37}$ one position towards the most signifiant bits (i.e., to the "left") because there is no X^7 term in $\texttt{37}$. (If there was a X^7 term, it would become X^8 after multiplication with X, hence an explicit modular reduction would be required to reduce the degree of the polynomial.)

- $\texttt{03} \times \texttt{21} = 0000\,0011 \times 0010\,0001$
  $= (X + 1) \times (X^5 + 1)$
  $= X^6 + X^5 + X + 1$
  $= 0110\,0011 = \texttt{63}$

Since $\texttt{03}$ equals $\texttt{01} + \texttt{02}$, we could have obtained the product by simply XORing the bitstring corresponding to $\texttt{21}$ with the same bitstring shifted by one position to the "left" (again, here because there is no X^8 or higher term that appears). We would indeed get

$$0010\,0001 \oplus 0100\,0010 = 0110\,0011$$

For the last two multiplications, since multiplying by $\texttt{01}$ is the identity, we immediately get

- $\texttt{01} \times \texttt{A5} = \texttt{A5} = 1010\,0101$
- $\texttt{01} \times \texttt{C0} = \texttt{C0} = 1100\,0000$.

Now that we have all four product, we can add them together (*i.e.* XOR them) to obtain b_0 :

$$\begin{aligned} b_0 &= 0110\,1110 \oplus 0110\,0011 \oplus 1010\,0101 \oplus 1100\,0000 \\ &= 0110\,1000 \\ &= \texttt{68} \end{aligned}$$

We now have the first coefficient of the output. All is left to do is perform the same operations for b_1, b_2 and b_3. For b_1, we would have to compute

$$b_1 = (\texttt{01} \times \texttt{37}) + (\texttt{02} \times \texttt{21}) + (\texttt{03} \times \texttt{A5}) + (\texttt{01} \times \texttt{C0})$$

After all the required computations, one would finally obtain

$$\begin{pmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{02} & \texttt{03} & \texttt{01} \\ \texttt{01} & \texttt{01} & \texttt{02} & \texttt{03} \\ \texttt{03} & \texttt{01} & \texttt{01} & \texttt{02} \end{pmatrix} \begin{pmatrix} \texttt{37} \\ \texttt{21} \\ \texttt{A5} \\ \texttt{C0} \end{pmatrix} = \begin{pmatrix} \texttt{68} \\ \texttt{41} \\ \texttt{1C} \\ \texttt{46} \end{pmatrix}$$

Exercise 7

Let $\mathbf{M}$ be the matrix of MixColumns. So a column of 4 bytes $(s_0, s_1, s_2, s_3)^\top$ is mapped to $(b_0, b_1, b_2, b_3)^\top = \mathbf{M}(s_0, s_1, s_2, s_3)^\top$ after MixColumns.

Because it is a linear operation, we have that $(b_0, b_1, b_2, b_3)^\top = \mathbf{M}(s_0, 0, 0, 0)^\top + \mathbf{M}(0, s_1, 0, 0)^\top + \mathbf{M}(0, 0, s_2, 0)^\top + \mathbf{M}(0, 0, 0, s_3)^\top$.

1. Using the property above, propose a way to implement MixColumns using only look-up's in precomputed tables and XORs. How many tables do you need? What is the size of each table? What do they contain?

2. Can you extend the reasoning so as to implement MixColumns $\circ$ SubBytes in a similar way? *Hint:* SubBytes processes each byte individually.

3. Can you find a variant with just one table? *Hint:* The matrix $\mathbf{M}$ has a special form.

Answer of exercise 7

1. We build four tables of 256 entries each, and each entry contains a 32-bit value. In the first table, we store $\mathbf{M}(s_0, 0, 0, 0)^\top$ for each value of s_0. In the second

table, we store $\mathbf{M}(0, s_1, 0, 0)^\top$ for each value of s_1, and similarly for the third and fourth tables.

To compute $\mathbf{M}(s_0, s_1, s_2, s_3)^\top$, we look up entry $\#s_0$ in the first table, entry $\#s_1$ in the second table, entry $\#s_2$ in the third table and entry $\#s_3$ in the fourth table, then we XOR all the results together.

2. Let $\mathsf{S}(x)$ be SubBytes applied to an individual byte x. What we need to compute now is $(b_0, b_1, b_2, b_3)^\top = \mathbf{M}(\mathsf{S}(s_0), \mathsf{S}(s_1), \mathsf{S}(s_2), \mathsf{S}(s_3))^\top$. The idea is the same as above, but now in the first table, we store $\mathbf{M}(\mathsf{S}(s_0), 0, 0, 0)^\top$ for each value of s_0, and similarly for the other tables.

3. Notice that $\mathbf{M}$ is a circulant matrix. So the entries in the second (third, fourth, resp.) table are equal to those in the first table after a shift by one (two, three, resp.) position(s). Hence, we keep only the first table and replace the look-up's in the other tables by a look-up in the first table followed by a shift.

Exercise 8

The Electronic Codebook (ECB) mode is a flawed encryption mode on top of a block cipher. What fundamental property makes it an inherently insecure mode?

How could an adversary easily win the IND-CPA game with just a few queries, even if he is not allowed to make queries with identical plaintext values? Detail the queries made by the adversary and the choice of (m_0, m_1).

Answer of exercise 8

The ECB mode is deterministic and its definition does not include a diversifier. Hence, fundamentally, ECB is not semantically or IND-CPA secure for this very reason.

Even if the adversary is restricted in that it cannot submit identical plaintexts, ECB has the property that identical blocks always encrypt to identical blocks. We exploit this property to win the IND-CPA game. Here is one possible way: The attacker chooses two arbitrary distinct strings A and B whose length is equal to the block size of the underlying block cipher. It queries the encryption of $A\|B$ and receives the ciphertext $X\|Y$, where X and Y are blocks. Then, it submits $m_0 = A$ and $m_1 = B$ to the challenger. If $c = X$, it means m_0 was encrypted; otherwise $c = Y$ and m_1 was encrypted. The adversary always wins.

Exercise 9

AES with CTR mode Suppose that a 160 byte message m was encrypted with AES-128 in a counter mode in order to obtain the ciphertext c. Let's suppose that the 10th byte of c was corrupted during a transmission.

a. Would the receiver be able to detect which byte was corrupted?

b. If we decrypt c, how many bytes of m would be affected by the error?

c. Does this number depend on the position where the error occurred?

Answer of exercise 9

a. No, the CTR mode does not provide authentication, nor any form of integrity protection, even non-cryptographic. So the receiver does not have any way to detect any corruption in the ciphertext.

b. 1 byte of 1 block.

c. No.

Exercise 10

AES with CBC mode Suppose that a 160 bytes message m was encrypted with AES-128 in CBC mode in order to obtain the ciphertext c. The block size is 16 bytes. Let's suppose that the 10th byte of c was corrupted during a transmission.

a. Would the receiver be able to detect which byte was corrupted?

b. If we decrypt c, how many bytes of m would be affected by the error?

c. Does this number depend on the position where the error occurred?

Answer of exercise 10

a. No, the CBC mode does not provide authentication, nor any form of integrity protection, even non-cryptographic. So the receiver does not have any way to detect any corruption in the ciphertext.

b. Two blocks would be affected by this modification: the 16 bytes of one block and only one byte of the next block, so 17 bytes in total.

c. Yes. If a byte of the last block is corrupted, then only the last block is affected.

Exercise 11

Consider the following encryption mode called DXCTR for "Diversifier-Xored CounTeR". It is a block cipher-based mode very similar to the original CTR, but the diversifier is XORed into the output of the block cipher instead of being part of its input. More precisely, DXCTR works as in Algorithm 1 and is illustrated in Figure 1. However, DXCTR is flawed.

1. What is/are intuitively the security problem(s) of DXCTR?
2. Describe how an adversary can win the IND-CPA game against DXCTR with only practical data and time complexities.
3. Why the original CTR mode does not have this problem?

Algorithm 1 The DXCTR encryption mode on top of block cipher E with block size n bits and key size m bits.

Input: secret key $K \in \mathbb{Z}_2^m$, plaintext $p \in \mathbb{Z}_2^*$ and diversifier $d \in \mathbb{N}$
Output: ciphertext $c \in \mathbb{Z}_2^{|p|}$

Cut p into blocks of n bits $(p_0, p_1, p_2, \ldots, p_l)$, except for the last one that can be shorter
Generate the keystream, for $i = 0$ to l

$$k_i = E_K(\text{block}(i)) \oplus \text{block}(d)$$

(here block(x) encodes the integer x into a string in $\mathbb{Z}_2^n$)
Truncate the last keystream block k_l to the size of p_l
Compute the ciphertext $c_i = k_i \oplus p_i$, for $i = 0$ to l

---

Answer of exercise 11

1. The problem here is that the diversifier does not diversify well enough the keystream. Since the diversifier is public and is only XORed, anyone can compute $c_i \oplus \text{block}(d) = E_K(\text{block}(i)) \oplus p_i$, which does not depend on the diversifier anymore. When the plaintext is known, this yields the sequence $E_K(\text{block}(i))$ that can then be used to decrypt other messages. When the plaintext is unknown, one can recover the difference between two different messages, say p

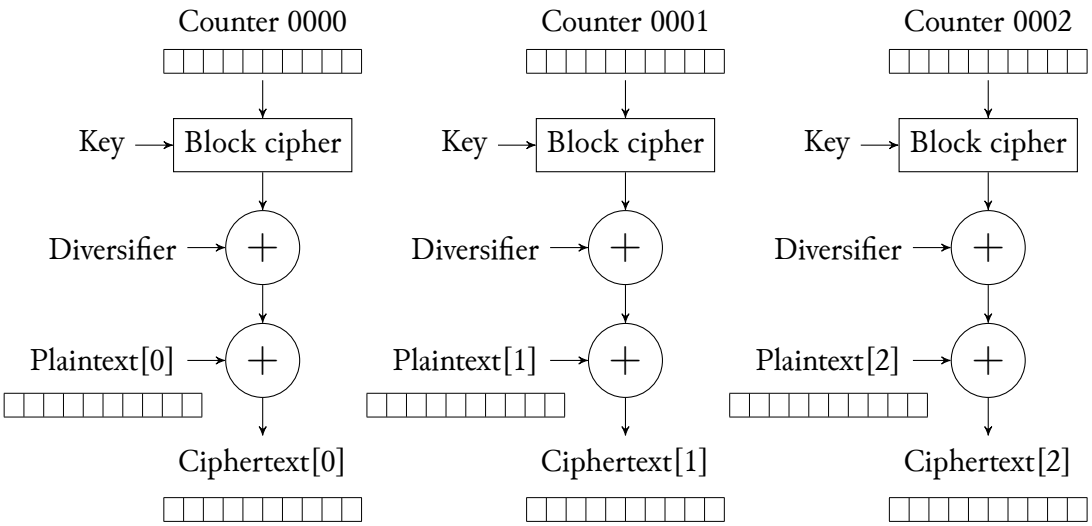

Figure 1: The DXCTR encryption mode.

and p', encrypted under different diversifiers d and d', respectively, by computing

$$c_i \oplus \text{block}(d) \oplus c'_i \oplus \text{block}(d') = p_i \oplus p'_i \ .$$

2. There are several ways to win the IND-CPA game, but the simplest one is probably to first determine a short sequence of $E_K(\text{block}(i))$ as explained above, then use it to decrypt the ciphertext returned by the challenger. For simplicity, we assume that the diversifier is a counter that is incremented at each encryption.

   (a) The challenger generates a secret key K, obviously unknown to the adversary.

   (b) The adversary queries $\text{DXCTR}_K(p, d)$ with $d = 1$ for an arbitrarily chosen plaintext p, and receives the corresponding ciphertext c. It then computes $c_i \oplus \text{block}(d) \oplus p_i = E_K(\text{block}(i))$.

   (c) The adversary arbitrarily chooses two different plaintexts p_0 and p_1 of the same length (as required by the game) that he sends to the challenger.

   (d) The challenger randomly chooses $b \in \{0, 1\}$, encrypts p_b with $d = 2$ and returns $c = \text{DXCTR}_K(p_b, d)$ the adversary.

   (e) The adversary deciphers c using the values of $E_K(\text{block}(i))$ computed at step 2 and determines whether p_0 or p_1 was chosen by the challenger. The adversary always wins despite the small number (1) of chosen-plaintext queries and the very limited amount of computations needed.

3. In CTR, the diversifier is used as input to the block cipher. The diversifier then has an unpredictable influence (for someone not knowing the key) on the keystream.

Exercise 12

Consider SHAKE128, a standard sponge function with permutation f, capacity $c = 256$ bits and rate $r = 1344$ bits. Let us assume that it is used in an application that uses it for the authentication of small messages.

In more details, the application computes a 128-bit MAC on a message M under the secret key K as SHAKE128$(K\|M)$. The secret key is 128-bit long and M is sufficiently short so that $K\|M$ fits in a single bock of r bits, even after padding[1].

1. Write symbolically, i.e., as a mathematical expression, the MAC as a function of the key and of the message for the restricted use case of this application.
2. Knowing that f and its inverse f^{-1} can both be evaluated efficiently, can the secret key be recovered from the value of the MAC? If not, please justify briefly. If so, please describe how.
3. Same question if we extend the MAC length to 1600 bits.
4. Same question if we extend the MAC length to 1600 bits and set $c = 0$, $r = 1600$.

Answer of exercise 12

1. After padding, the first (and only) block of input is $K\|M\|1\|0^{r-|M|-|K|-1}$. The number of zeroes is such that the block is r-bit long. After adding the block to the outer part, the state is $K\|M\|1\|0^{r-|M|-|K|-1+c}$, i.e., the block followed by c additional zero bits at the end. After applying the permutation f, the state is now $f(K\|M\|1\|0^{r-|M|-|K|-1+c})$.

   So the MAC is $f(K\|M\|1\|0^{r-|M|-|K|-1+c})$ truncated to the first 128 bits.

2. No, the key cannot be recovered.

   Even if we can compute f^{-1} efficiently, the MAC value reveals only the first 128 bits of $f(K\|M\|1\|0^{r-|M|-|K|-1+c})$. Exhaustively searching through the $1600 - 128 = 1472$ remaining bits would take much more time than doing an exhaustive key search.

[1] For simplicity, we assume that the input $K\|M$ is simply padded as $K\|M\|10^*$, i.e., with a single bit 1 followed by the minimum number of bits 0 such that the padded message is r-bit long.

3. No, the key cannot be recovered either.

   Remember that the output is squeezed out in blocks of $r = 1344$ bits. If we extend the MAC length to 1600 bits, the output will be given in two iterations, first 1344 bits, then the remaining 256 bits after an iteration of f. The first MAC block reveals the first 1344 bits of $f(K\|M\|1\|0^{r-|M|-|K|-1+c})$, but again exhaustively searching through the $1600 - 1344 = 256$ remaining bits would take much more time than doing an exhaustive key search.

4. Yes, in this case, the key can be recovered.

   Now the MAC is given in one iteration of $r = 1600$ bits, revealing the entire state, i.e., the entire value of $f(K\|M\|1\|0^{r-|M|-|K|-1+c})$. Computing f^{-1} on it and extracting the first 128 bits yields K.

   This highlights that a (keyed) sponge function with $c = 0$ has no security.

Design of symmetric primitives

Exercise 13

Design of symmetric crypto primitives Below are the specifications of a cryptographic permutation under development, called ABCD. We first give the specifications of the permutation, then ask you some questions about it.

ABCD is a 256-bit permutation. The authors lacked inspirations and gave it its name because the 256-bit input block and running state are represented as 4 words (or rows) of 64 bits each, denoted a, b, c and d. We denote the individual bits in a rows using subscripts, e.g., row a is composed of the bits $a_0, a_1, \ldots, a_{63}$. The four bits (a_i, b_i, c_i, d_i) at the same position in each row is called a column. In other words, the state can be viewed as a grid of 4 rows by 64 columns.

The evaluation of the permutation consists in 13 rounds. The evaluation of permutation goes as follows:

for each round $r = 0$ to 12 **do**
  ShakeColumns
  PreShiftRows
  StirColumns
  PostShiftRows
  AddRoundConstant(r)

We now describe the five step mappings AddRoundConstant, ShakeColumns, PreShiftRows, StirColumns and PostShiftRows, all specified at the bit level, or more formally, in the Galois field $\mathrm{GF}(2) = \{0, 1\}$. In this field, $x + y$ (resp. xy) denotes the modulo-2 addition (resp. multiplication) of $x, y \in \mathrm{GF}(2)$. We use the operator $\oplus$ to express the component-wise addition of vectors of elements in $\mathrm{GF}(2)$, such as rows, columns or states.

Definition of AddRoundConstant(r)

for each column $i = 0$ to 63 **do**
  if $i \leq r$ **then**
    $a_i \leftarrow a_i + 1$

Definition of ShakeColumns

for each column $i = 0$ to 63 **do**
  $p \leftarrow a_i + b_i + c_i + d_i$
  $a_i \leftarrow a_i + p$
  $b_i \leftarrow b_i + p$
  $c_i \leftarrow c_i + p$
  $d_i \leftarrow d_i + p$

Definition of PreShiftRows

$a \leftarrow a$
$b \leftarrow \mathrm{rot}^1(b)$
$c \leftarrow \mathrm{rot}^3(c)$
$d \leftarrow \mathrm{rot}^9(d)$

For a row x, $\mathrm{rot}(x)$ denotes the operation that consists in cyclically shifting all the bits by one position, i.e.,

$$(x_0, x_1, x_2, \ldots, x_{63}) \rightarrow (x_1, x_2, \ldots, x_{63}, x_0).$$

Definition of StirColumns

for each column $i = 0$ to 63 **do**
  $a_i \leftarrow a_i + b_i c_i$
  $b_i \leftarrow b_i + c_i d_i$
  $c_i \leftarrow c_i + d_i a_i$
  $d_i \leftarrow d_i + a_i b_i$

Definition of PostShiftRows

$a \leftarrow a$

$b \leftarrow \text{rot}^1(b)$
$c \leftarrow \text{rot}^5(c)$
$d \leftarrow \text{rot}^{25}(d)$

A: Classify each step mapping as linear, affine or non-linear. For affine or linear mappings, please provide a short justification (one line). For non-linear mappings, please provide an example of input pair that violates the definition of affinity.

B: Point out the step mapping(s) that provide diffusion, with a short justification.

C: For the step mapping ShakeColumns, please explain what happens at the output when:

- one flips 1 bit at the input;
- one flips 2 bits of the same column at the input;
- one flips 2 bits of different columns at the input.

D: Write the specifications of the inverse of ABCD.

E: For each the five step mappings AddRoundConstant, ShakeColumns, PreShiftRows, StirColumns and PostShiftRows, would you characterize it as *bend*, *mix*, *notch* or *shuffle*?

F: Without AddRoundConstant, ABCD would satisfy an (undesired) symmetry property. Which one is it? By symmetry property, we are looking for an invertible operation S on 256-bit strings that commutes with ABCD, i.e., S is such that $\forall x \in \{0,1\}^{256}$, we have $\text{ABCD}(S(x)) = S(\text{ABCD}(x))$.

Answer of exercise 13

A:

- AddRoundConstant is affine. It is made of additions with a constant.
- ShakeColumns is linear. Each bit is the sum of input bits, and there are no additions with a constant (i.e., $\text{ShakeColumns}(0) = 0$).

- PreShiftRows and PostShiftRows are linear. These are bit transpositions, hence linear.
- StirColumns is non-linear. The use of multiplication gives a hint. Since StirColumns consists in the application of 64 identical mappings on each column, we can give a counterexample on a single column. Using Table 1, we see that

$$\begin{aligned}\mathsf{StirColumn}(0001) \oplus \mathsf{StirColumn}(0010) &= 0011 \\ &\neq 0111 = \mathsf{StirColumn}(0001 \oplus 0010).\end{aligned}$$

B: Only ShakeColumns and StirColumns provide diffusion. Clearly, each input bit influences more than one output bit.

PreShiftRows and PostShiftRows are bit transpositions, hence each input bit only influences exactly one output bit. In AddRoundConstant too, each input bit only influences exactly one output bit.

C:

- When one flips 1 bit at the input, in that column, the three other bits of the same column flip at the output. Let us assume that a_i is flipped at the input. Then, the value of p flips, and b_i, c_i and d_i flip at the output. The output a_i does not flip, as $a_i \leftarrow (a_i + 1) + (p + 1) = a_i + p$. (The reasoning is similar when b_i, c_i or d_i is flipped at the input.)
- When one flips 2 bits of the same column at the input, in that column, only these two bits flip at the output. When two bits are flipped in the same column, the value of p is unchanged since $1 + 1 = 0$. Hence, only the same two bits flip at the output.
- When one flips 2 bits of different columns at the input, the three other bits of these two columns flip at the output. The reasoning is the same as that of the first bullet, but on two columns in parallel, since ShakeColumns works on columns independently.

D: Each operation is inverted, starting from the last one. One can notice that AddRoundConstant(r) is clearly self-inverse. ShakeColumns is self-inverse too, as detailed below. Hence, the inverse of ABCD works like this:

for each round $r = 12$ down to 0 **do**

AddRoundConstant(r)
InvPostShiftRows
InvStirColumns
InvPreShiftRows
ShakeColumns

To see that ShakeColumns is self-inverse, notice that p, the parity of a column, is unchanged by the application of ShakeColumns. Hence, to undo the effect of ShakeColumns, it is sufficient to add again the parity of each column.

The inverses of PostShiftRows and PreShiftRows are obtained by applying the cyclic shift in the other direction.

Notice that StirColumns works as a Feistel network. To undo it, it suffices to run it backwards:

Definition of InvStirColumns
for each column $i = 0$ to 63 **do**
$d_i \leftarrow d_i + a_i b_i$
$c_i \leftarrow c_i + d_i a_i$
$b_i \leftarrow b_i + c_i d_i$
$a_i \leftarrow a_i + b_i c_i$

E:

- *bend*: StirColumns provides non-linearity over a column;
- *mix*: ShakeColumns provides good diffusion over a column;
- *notch*: AddRoundConstant breaks the symmetry of translation along the rows (see below);
- *shuffle*: PreShiftRows and PostShiftRows are bit transpositions that ensure that the four bits of a column are moved to different columns before the next StirColumns or ShakeColumns. Hence they ensure the global diffusion and the global spread of non-linearity as more rounds are applied.

F: Without AddRoundConstant, all the operations are invariant under a cyclic translation of all the rows by the same amount. Hence, without AddRoundConstant, the permutation ABCD would have this symmetry property, i.e., shifting the whole input along the rows would just shift the whole output.

More formally, an invertible operation S that would commute with ABCD would be:

$$S(a, b, c, d) = (\mathrm{rot}(a), \mathrm{rot}(b), \mathrm{rot}(c), \mathrm{rot}(d)).$$

With AddRoundConstant, if one shifts the whole input of ABCD along the rows, we can expect that the output will change in a complicated way.

$a_i b_i c_i d_i$	$a'_i b'_i c'_i d'_i = \mathsf{StirColumn}(a_i b_i c_i d_i)$
0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 1
0 0 1 0	0 0 1 0
0 0 1 1	0 1 1 1
0 1 0 0	0 1 0 0
0 1 0 1	0 1 0 1
0 1 1 0	1 1 1 1
0 1 1 1	1 0 0 1
1 0 0 0	1 0 0 0
1 0 0 1	1 0 1 1
1 0 1 0	1 0 1 0
1 0 1 1	1 1 0 0
1 1 0 0	1 1 0 1
1 1 0 1	1 1 1 0
1 1 1 0	0 1 1 0
1 1 1 1	0 0 1 1

Table 1: Truth table of StirColumns applied to a single column.

Université Libre de Bruxelles
Faculté des Sciences

2022-2023

INFO-F-405: Introduction to cryptography

3. Hashing

Definitions and requirements

Exercise 1

For a given hash function, does second preimage resistance imply collision resistance, or vice-versa, and why?

Answer of exercise 1

Collision resistance implies second preimage resistance (and not vice-versa).

Let us consider an adversary who is able to find a second preimage of $y = \text{hash}(x)$, i.e., he is able to find $x' \neq x$ such that $\text{hash}(x') = y = \text{hash}(x)$. By definition of the second preimage, he is given the value x. This means he is also able to produce a collision since $x' \neq x$ and $\text{hash}(x') = \text{hash}(x)$. So, an second preimage attack implies a collision attack.

By contraposition, assume that a hash function is collision-resistant, i.e., it is infeasible to find a collision. Then, if an adversary could find a second preimage, he would also be able find a collision, which was assumed to be infeasible. Hence, it is also infeasible to find a second preimage. Therefore, collision resistance implies second preimage resistance.

If a hash function is second preimage resistance, i.e., it is infeasible to find a second preimage, a collision could nevertheless be found by other means than via a second preimage attack. Intuitively, a collision attack does not require a specific image y, so a collision attack in general does not help finding a preimage for the given image y. Therefore, collision resistance and second preimage resistance are not equivalent.

Exercise 2

Choose a standard cryptographic hash function (or XOF) arbitrarily. Write a small program that computes the digest of a given string truncated to n bytes. For a given

value n, compute the digest of arbitrary (but distinct) input strings, and look for a collision at the output.

1. Start with $n = 1$, so $\ell = 8$ bits of output. After how many iterations t do you get a collision?

2. Increase n until this starts to take too much time. Do you recognize the law that gives you $t(n)$?

Answer of exercise 2

You should be able to recognize the birthday paradox with $t \approx 2^{\ell/2} = 2^{4n}$.

Exercise 3

Use the previously written piece of code and do the following test. Choose an arbitrary message m and compute $H = \text{hash}(m)$. Then, flip the first bit of m in order to obtain m'. Can you predict the value of $H' = \text{hash}(m')$ from that of H without explicitly computing it?

Repeat the experiment with different values of m and/or flipping bits at different positions. Can you see a predictable relationship between the corresponding H and H'? What is the average number of differences (i.e., the Hamming distance) between H and H'?

Answer of exercise 3

No, H and H' should have no apparent relationship, even if m and m' differ only in one bit position.

The average of the Hamming distance between H and H' should be about $\ell/2$.

Exercise 4

A friend of yours designed his/her own hash function CATSHA320, with 320 bits of output. It is an iterated hash function that cuts the input message into blocks of 192 bits, processes them sequentially and has a chaining value of 224 bits. Without knowing more about this hash function, what is its expected collision resistance (in number of attempts), and why?

Same question for CATSHA320++ with the same specifications but a block size of 384 bits and a chaining value of 480 bits.

Answer of exercise 4

The adversary can either find an output collision or an internal collision.

In the former case, considering CATSHA320 as a black box, we can expect to find a collision after about $\sqrt{2^{320}} = 2^{160}$ attempts, following the birthday paradox. In the latter case, the chaining value after processing some input blocks can itself be viewed as a black box. Since it has 224 bits, we can expect to find a collision after about $\sqrt{2^{224}} = 2^{112}$ attempts, following the birthday paradox. Hence, the weakest link is the internal collision, and CATSHA320 cannot have more than 112 bits of collision resistance, i.e., after 2^{112} attempts.

For CATSHA320++ with a chaining value of 480 bits, the weakest link is now the output collision, and CATSHA320++ cannot have more than 160 bits of collision resistance, i.e., after 2^{160} attempts.

Note that the block size does not play a role in this exercise. (The only role the block size could play is in the number of blocks required for each attempt to have enough degrees of freedom. In CATSHA320, the 192 bits of a single block are enough to generate 2^{112} different messages for the 2^{112} attempts necessary to get the internal collision. Hence the adversary can generate attempts using 1-block messages. If the block size was, for instance, 16 bits, then at least 7 blocks of messages are necessary to be able to generate 2^{112} different messages.)

Zooming from Merkle-Damgård into SHA-1

Exercise 5

[1] The core of some hash functions is the compression function. Let assume that we build a compression function on top of the block cipher

$$\text{output block} = \text{AES}_{\text{key}}(\text{input block})$$

in one of the following ways:

- $f_1(x, y) = \text{AES}_x(y) \oplus x$
- $f_2(x, y) = \text{AES}_x(x) \oplus y$

We ask you to find collisions for f_1 and for f_2. In other words you should find values a_1, b_1 and a_2, b_2 such that $f_1(a_1, b_1) = f_1(a_2, b_2)$ and also find values c_1, d_1 and c_2, d_2 such that $f_2(c_1, d_1) = f_2(c_2, d_2)$.

Answer of exercise 5

[1] This exercise was inspired by an exercise from the Security course given by Dan Boney.

- For f_1: find x, y, x', y' such that $\text{AES}_x(y) \oplus x = \text{AES}_{x'}(y') \oplus x'$. Pick x, y, x' at random and set $y' = \text{AES}_{x'}^{-1}(\text{AES}_x(y) \oplus x \oplus x')$. We can invert AES because we know x'.
- For f_2: find x, y, x', y' such that $\text{AES}_x(x) \oplus y = \text{AES}_{x'}(x') \oplus y'$. Pick x, x' at random and set $y = \text{AES}_{x'}(x')$ and $y' = \text{AES}_x(x)$

Exercise 6

Many well-known hash functions, such as MD5, SHA-1 and the SHA-2 family, use a block cipher in the Davies-Meyer construction. However, these are all specific, dedicated, block ciphers. What about the AES? So let us build SHAES by first building a compression function from the AES-256 in the Davies-Meyer construction, then putting the compression function in the Merkle-Damgård construction.

1. What is the block size of SHAES?
2. What is the chaining value size of SHAES?
3. Is SHAES a valid solution in practice?

Answer of exercise 6

1. As it uses Davies-Meyer, the block size of SHAES is equal to the key size of AES-256, that is, 256 bits.
2. As it uses Davies-Meyer, the chaining value size of SHAES is equal to the block size of AES-256, that is, 128 bits.
3. With only 128 bits of chaining value, SHAES is vulnerable to internal collisions with a complexity of only 2^{64} attempts. This is weak in today's standards.

Exercise 7

Some designs, like the hash function SHA-1 and the SHA-2 family, use a combination of modular addition (in $\mathbb{Z}_{2^n}$) and bitwise addition or XOR (in $\mathbb{Z}_2^n$), plus rotations (or circular shifts). In the litterature, this combination is often referred to "ARX" for addition-rotation-XOR. The idea is that the modular addition is non-linear from the point of view of $\mathbb{Z}_2^n$, and vice-versa, the XOR is non-linear in $\mathbb{Z}_{2^n}$.

Formally, an element $x = (x_0, x_1, \ldots, x_{n-1}) \in \mathbb{Z}_2^n$ is mapped to the integer $X = \sum_i x_i 2^i$ in $\mathbb{Z}_{2^n}$. Let us define $X = \mathsf{integer}(x)$ as the function that converts an element of $\mathbb{Z}_2^n$ to an element of $\mathbb{Z}_{2^n}$, and let $x = \mathsf{vector}(X)$ be its inverse.

1. Show that $f(x, y) = \mathsf{vector}(\mathsf{integer}(x) + \mathsf{integer}(y))$, i.e., the modular addition of x and y interpreted as integers, is non-linear in $\mathbb{Z}_2^n$. *Hint:* It is sufficient to give a counter-example. Also, you may set $n = 2$ for simplicity.

2. Show that $F(X, Y) = \mathsf{integer}(\mathsf{vector}(X) \oplus \mathsf{vector}(Y))$, i.e., the XOR of X and Y interpreted as n-bit vectors, is non-linear in $\mathbb{Z}_{2^n}$.

Answer of exercise 7

1. Let $n = 2$. Note that $f(0, 0) = 0$, so we have to show that $f(x \oplus x', y \oplus y') \neq f(x, y) \oplus f(x', y')$ for some counterexample. Take $(x, y) = (1, 0)$ and $(x', y') = (0, 1)$. Then,

$$f(1, 1) = 2 \quad \neq \quad f(1, 0) \oplus f(0, 1) = 1 \oplus 1 = 0 \ .$$

2. Similarly, we have to show that $f(X + X', Y + Y') \neq F(X, Y) + F(X', Y')$ for some counterexample. Take $(X, Y) = (1, 0)$ and $(X', Y') = (0, 1)$. Then,

$$F(1, 1) = 0 \quad \neq \quad F(1, 0) + F(0, 1) = 1 + 1 = 2 \ .$$

Modern generic security

Exercise 8

Let $\mathcal{RO}(x)$ be a random oracle that outputs an infinite sequence of uniformly and independently distributed bits for each input $x \in \{0, 1\}^*$. If the random oracle is queried twice with the same input, it always returns the same sequence. We denote with $\mathcal{RO}(x, \ell)$ the output of $\mathcal{RO}(x)$ truncated after the first ℓ output bits, so $\mathcal{RO}(x, \ell) \in \{0, 1\}^\ell$.

1. *Preimage attack.* Given some value $y \in \{0, 1\}^\ell$, an adversary would like to find a preimage, i.e., an input x such that $\mathcal{RO}(x, \ell) = y$. What is the probability of success after t attempts? (We can approximate assuming that $t \ll 2^\ell$.)

2. *Multi-target preimage attack.* Given a set of m distinct values $\{y_1, \ldots, y_m\}$, with $y_i \in \{0, 1\}^\ell$, an adversary would be happy to find a preimage of any one of these images. What is the probability of success after t attempts?

Let $h(x)$ be a concrete extendable output function (XOF) that outputs a potentially infinite sequence of bits, and we similarly denote $h(x, \ell)$ its truncation to ℓ output bits. As of today, the only known attack against $h(x)$ has a probability of success $t/2^{c/2}$, for some security parameter c, and where t is the time spent by the adversary expressed in the number of attempts. This means that, except for this probability $\epsilon(t)$, we can model $h(x)$ as a random oracle.

3. What is the range of output sizes ℓ and parameter values c such that $h(x)$ offers 128 bits of preimage resistance?

4. What is the range of output sizes ℓ and parameter values c such that $h(x)$ offers 128 bits of multi-target preimage resistance with $m = 2^{32}$?

You may use the approximation $\log(a + b) \approx \max(\log a, \log b)$.

Answer of exercise 8

1. The probability that an adversary finds x such that $\mathcal{RO}(x, \ell) = y$ after t attempts is approximately $t2^{-\ell}$.

   After one attempt, the probability that $\mathcal{RO}(x, \ell) = y$ is $2^{-\ell}$, as the range of $\mathcal{RO}(x, \ell)$ has a size of 2^{ℓ}. It is tempting to say that after t attempts, the probability will be $t2^{-\ell}$, but this would be rigorously correct in the case of drawing without replacement. However, this remains approximately true when $t \ll 2^{\ell}$. To see this, let us consider the probability that the correct output does *not* appear after t attempts: $(1 - 2^{-\ell})^t$. Using binomial expansion, we see that $(1 - 2^{-\ell})^t = 1 - t2^{-\ell} + o(t2^{-\ell})$.

2. Similarly, the probability is approximately $mt2^{-\ell}$ since each attempt has a m chances out of 2^{ℓ} to hit one of the y_i's.

3. We are not given any information about the known attack against h, so we have to consider the worst case that this attack helps find the preimage. After t attempts the adversary can find the preimage $h(x) = y$ either by chance (i.e., $t2^{-\ell}$) or because it results from the attack on h (i.e., $t2^{-c/2}$). The probability of the union of two events is upper bounded by the sum of the probability of the two events, i.e., $\Pr[A \text{ or } B] \leq \Pr[A] + \Pr[B]$, so we have to consider as success probability $\epsilon(t) = t2^{-\ell} + t2^{-c/2}$. Working this out:

$$\begin{aligned} & t2^{-\ell} + t2^{-c/2} \\ =& t(2^{-\ell} + 2^{-c/2}) \\ \approx& t2^{\max(-\ell, -c/2)} \\ =& t2^{-\min(\ell, c/2)}. \end{aligned}$$

In order to have 128 bits of security, we need to ensure that $t2^{-\min(\ell,c/2)}$ does not reach 1 before $t = 2^{128}$. Therefore, we must have $\ell \geq 128$ and $c/2 \geq 128$ or, equivalently, $c \geq 256$.

4. The reasoning is similar, but this time with $\epsilon(t) = mt2^{-\ell} + t2^{-c/2}$. Working this out:

$$\begin{aligned} & mt2^{-\ell} + t2^{-c/2} \\ =& t(2^{-\ell+\log_2 m} + 2^{-c/2}) \\ \approx& t2^{\max(-\ell+\log_2 m, -c/2)} \\ =& t2^{-\min(\ell-\log_2 m, c/2)}. \end{aligned}$$

In order to have 128 bits of security, we must have $\ell - \log_2 m \geq 128 \Leftrightarrow \ell \geq 128 + 32 = 160$ and $c/2 \geq 128 \Leftrightarrow c \geq 256$.

Inside SHA-3

Exercise 9

Let π be a cryptographically secure permutation that maps $b = 1600$-bit strings onto b-bit strings. We assume that π and its inverse π^{-1} are both equally efficiently implementable. We build a hash function by applying the sponge construction on top of π. The hash function has a fixed output length of $n = 256$ bits. However, for the sake of this question, we set the capacity to $c = 0$.

1. Please explain how to obtain a collision with this hash function.
2. Given two prefixes x and y of b bits each, show how to obtain a collision between one input that must start with x and the other one with y.
3. Given an output value z of n bits, describe how to find a preimage with this hash function.
4. Now let us assume that this hash function is used exclusively in an application that hashes input strings made of ASCII-encoded text, for which each byte has its most significant bit always set to 0. More precisely, we restrict the input strings m such that every 8th bit is 0, and for the sake of simplicity this is the only restriction we consider (i.e., we do not care whether m is actually valid ASCII-encoded text or not). Please explain how to obtain a collision with this hash function such that the colliding inputs have this restriction. What is the complexity of your attack?

5. Given an output value z of n bits, describe how to find a preimage with this hash function such that the preimage has the aforementioned restriction. What is the complexity of your attack?

Answer of exercise 9

1. When $c = 0$, the sponge construction claims no security, but how can we obtain a collision? The simplest way consists in obtaining a state collision, i.e., two different sequences of input blocks that lead to the same value of the state. Because there is no inner part ($c = 0$), we can control the entire state value by appropriately choosing the input blocks as they are XORed in, and we can easily set the state to an arbitrarily chosen value.

   With this in mind, there are numerous ways to make a collision. The simplest consists in resetting the state to its initial value 0^b. Let $\mathsf{tozero} = \pi^{-1}(0^b)$. Then, after absorbing tozero the state is back to 0^b. Hence, for any arbitrary string s, we have $h(\mathsf{tozero}|s) = h(s)$.

2. After absorbing as first block x (resp. y), the state is $\pi(x)$ (resp. $\pi(y)$). We need to find one block x' to put after x and one block y' to put after y so that the same state is reached after they are XORed in. In other words,

   $$\pi(x) \oplus x' = \pi(y) \oplus y' \ .$$

   Any choice such that $x' \oplus y' = \pi(x) \oplus \pi(y)$ is suitable, for instance $(x', y') = (\pi(x), \pi(y))$ or $(x', y') = (\pi(y), \pi(x))$. Then, we have $h(x|x') = h(y|y')$.

3. We extend z by appending $1600 - 256$ arbitrary bits z' and consider this as the last state of the sponge function, $s_1 = z|z'$. We then compute $s_0 = \pi^{-1}(s_1)$, the state after padding and absorbing the first block. We unpad s_0 by removing the last bits 0 until a 1 is found and removed. This works only if $s_0 \neq 0^b$; in the unlikely case where $s_0 = 0^b$, we just restart with another arbitrary value z'.

4. When the input is restricted and every 8th bit must be set to zero, the adversary cannot directly influence the value of the state in those bits, exactly as for the inner part. There are $b/8 = 200$ bits that are restricted, and hence the behavior is identical to a sponge function with an inner part of $c = 200$ bits, except for the location of this inner part. In the sequel, let us call the *virtual inner part* every 8th bit of the state that coincides with the input bits that are stuck to 0.

   To get a collision in the state, we start with finding two input blocks x and y (satisfying the restriction) such that, after being absorbed, the value of the

virtual inner part is identical. With these, the state is $\pi(x)$ and $\pi(y)$, respectively. Then we proceed as in the sub-question above, i.e., we find two next input blocks x' and y' that cancel the difference between $\pi(x)$ and $\pi(y)$. Because the virtual inner part is identical, x' and y' can satisfy the restriction. Then, we have $h(x|x') = h(y|y')$.

The time-consuming part is to find the collision in the virtual inner part. Because of the birthday paradox, this takes about $\sqrt{2^{200}} = 2^{100}$ attempts.

5. Here we can proceed as in sub-question 3, except that the state $s_0 = \pi^{-1}(z|z')$ must have the virtual inner part equal to zero for the input to satisfy the restriction. A simple attack consists in trying different values z' until this happens. For every attempt, the probability is about 2^{-200}, so the complexity is about 2^{200}.

   (For completeness #1, the unpadding must also succeed. This typically requires that the last byte of s_0 contains $(10000000)_2$, which adds 7 more restrictions, for a total complexity of 2^{207}.)

   (For completeness #2, there exist more advanced preimage attacks on sponge functions that exploit the birthday paradox, and they can be transposed to this case. Using these, the complexity therefore drops to around 2^{100}.)

Exercise 10

Consider FIPS 202 and extendable output functions.

1. What is (approximately) the performance ratio between SHAKE128 and SHAKE256? And between SHAKE128 and SHA3-512?

2. For SHAKE128, what is the output size you need to choose to have 128-bit security for collision, preimage and second preimage resistance?

3. Same question, but only for preimage and second preimage resistance (no collision resistance required)?

4. Let us fix an ouput size ℓ. Given a set of m distinct values $\{y_1, \ldots, y_m\}$, with $y_i \in \{0,1\}^\ell$, an adversary would be happy to find a preimage of any one of these images. What is (approximately) the probability of success after t random attempts against a random oracle?

5. Say that $m = 2^{40}$. Can we use SHAKE128 to resist against multi-target attacks with a security level of 128 bits? If so, what would be the required output size ℓ?

Answer of exercise 10

1. We need to compare the corresponding rates. SHAKE128 is based on Keccak with $c = 256$, SHAKE256 on Keccak with $c = 512$, and SHA3-512 has $c = 1024$[2]; and the rate r is given by the formula $r = b - c$ with b=1600, the usual value for Keccak.
   As a result, for SHAKE128, $r = 1600 - 256 = 1344$, for SHAKE256, $r = 1600 - 512 = 1088$ and for SHA3-512, $r = 1600 - 576$.
   We can now compute the rates. SHAKE128 is $\frac{1344}{1088}$ times faster than SHAKE256 and $\frac{1344}{576}$ times faster than SHA3-512.

2. We know that preimage resistance is $2^{\min(n,c/2)}$, second-preimage resistance is $2^{\min(n,c/2)}$ and collision resistance is $2^{\min(n/2,c/2)}$ for the sponge construction[3].
   We can observe that the collision resistance is the critical one among the three $2^{\min(n/2,c/2)}$ bits of security. Because of the birthday paradox, we need 256 bits of output to reach 128 bits of collision resistance.

3. Here 128 bits of output are sufficient.

4. One attempt has a probability of $m2^{-\ell}$ of hitting one of the m given images. After $t \ll \frac{2^\ell}{m}$ attempts, the probability is approximately t times greater, hence $mt2^{-\ell}$. So an adversary would need about $t \approx \frac{2^\ell}{m} = 2^{\ell - \log_2 m}$ attempts to find one of the preimages. Hence, for the same output length, finding a multi-target preimage is about $\log_2 m$ bits easier than a regular (single-target) preimage attack.

5. Yes. SHAKE128 should resist against any attack up to complexity 2^{128} unless easier on a random oracle. So if we can find a solution that achieves 128 bits of security for the random oracle, the same applies to SHAKE128. (If the question was on more than 128 bits of security, the answer would have been no.) So, if we output at least $\ell = 128 + \log_2 m = 168$ bits, SHAKE128 can still achieve 128 bits of security against this attack.

Others

[2] See slide set 3-Hashing, page 43.
[3] From slide 31 of the slide set "3-Hashing"

Exercise 11

How (not?) to build message authentication codes from hash functions.

Let K be a k-bit secret key (e.g., $k = 128$ bits).

1. Let us assume that we build a MAC function as follows:

   $$\mathrm{MAC}_K(\text{message}) = \mathrm{hash}(\text{message}) \oplus K,$$

   where $\mathrm{hash}(\cdot)$ is a secure k-bit hash function and $\oplus$ denotes the bitwise mod-2 addition (or XOR). Is this MAC function secure? If so, please justify briefly. If not, please describe an attack in the light of the EU-CMA game.

2. Let us consider another MAC function. We now assume that the MAC function is constructed as follows:

   $$\mathrm{MAC}_K(\text{message}) = \mathrm{hash}(K_1 \| \text{message}) \oplus K,$$

   where $\mathrm{hash}(\cdot)$ is a secure k-bit hash function, $\|$ denotes the concatenation and $K = K_1 \| K_2$ with $|K_1| = |K_2| = k/2$. What is the security strength of this MAC function, and why?

Answer of exercise 11

1. No, it is not secure. From a single known (message, tag) pair, the attacker can recover the key K by computing $K = \mathrm{hash}(\text{message}) \oplus \text{tag}$.

   In the language of the EU-CMA game, the adversary can produce a forgery with probability 1 by doing the following:

   - The adversary queries $\text{tag} = \mathrm{MAC}_K(\text{message})$ with an arbitrary message. From this, K is recovered as explained above.
   - Knowing the key, the adversary computes $\text{tag}' = \mathrm{MAC}_K(\text{message}')$ with a different message.
   - The adversary wins as $(\text{message}', \text{tag}')$ is a valid pair that was not queried before.

2. The security strength is $k/2$ bits. The attack works similarly, except that we need to exhaustively search for the correct value of K_1, which means a time complexity of $2^{k/2}$. In more details, the attack works as follows.

   - Get a known (message, tag) pair by querying $\text{tag} = \mathrm{MAC}_K(\text{message})$ with an arbitrary message.

- For each candidate value K_1^* for the first $k/2$ bits of K:
  - Compute $K^* = \text{hash}(K_1^* \| \text{message}) \oplus \text{tag}$.
  - If the first $k/2$ bits of K^* do not match K_1^*, this cannot be a correct guess for the candidate K_1^*, so we continue. Otherwise, if they match, K^* might be the correct key; we double-check with another (message, tag) pair; if correct, we found the secret key K.

Université Libre de Bruxelles
Faculté des Sciences

2023-2024

INFO-F-405: Introduction to cryptography

Introduction to modular arithmetic

Theoretical background

Euler φ function

The Euler φ function gives the number of integers between 0 and $n-1$ coprime to n. For example, $\varphi(20) = 8$ because only the 8 integers $\{1, 3, 7, 9, 11, 13, 17, 19\}$ are coprime to 20.

A direct consequence of this theorem is that for any p, a prime number, $\varphi(p) = p-1$. More generally, $\varphi(p^m) = p^m - p^{m-1} = (p-1) \cdot p^{m-1}$.

Let us also note this property of φ that if $gcd(m, n) = 1$, then $\varphi(m \cdot n) = \varphi(m) \cdot \varphi(n)$.

As a result, it is easy to compute $\varphi(n)$ when we know the prime factors factorization of n. Indeed, if $n = p_1^{m_1} \cdot p_2^{m_2} \cdots p_v^{m_v}$, with all the p_i prime numbers, we have:

$$\varphi(n) = (p_1 - 1)p_1^{m_1-1}(p_2 - 1)p_2^{m_2-1} \cdots (p_v - 1)p_v^{m_v-1} \tag{1}$$

For example $20 = 2^2 \cdot 5$ and $\varphi(20) = (2-1) \cdot 2 \cdot (5-1) = 8$

Additive structure of multiplication

For modulus n of the form p^k, $2p^k$ where p is a prime and $k > 0$, there exists an integer g (called the generator) such that the set of powers of g, $\{g^0, g^1, g^2, \cdots, g^{\varphi(n)-1}\}$ is the set of all integers coprime to n.
For example, if $n = 10$, we have $g = 3$ and $\{1, 3, 9, 27\} \equiv \{1, 3, 7, 9\}$.

Furthermore, $g^{\varphi(n)} \equiv 1 \equiv g^0$, meaning that the exponents of g can be reduced mod $\varphi(n)$. If we multiply two integers $a = g^\alpha$ and $b = g^\beta \bmod n$, their exponents add mod $\varphi(n)$: $ab = g^\alpha g^\beta = g^{(\alpha+\beta) \bmod \varphi(n)}$.
For example, modulo 10, $7 \equiv 3^3$ and $9 \equiv 3^2$, hence $7 \cdot 9 = 3^{3+2} \equiv 3^1 = 3$ because $\varphi(10) = 4$.

To compute the multiplicative inverse of an integer $a = g^\alpha \bmod n$, one can simply take the additive inverse of the exponent mod $\varphi(n)$. Hence $a^{-1} \equiv g^{(-\alpha) \bmod \varphi(n)}$

Modular exponentiation

Modular exponentiation is the computation of $a^b \bmod n$. Working modulo n, if we have a generator g and $a \equiv g^\alpha$, to compute a^b, one can simply compute $(g^\alpha)^b = g^{\alpha \cdot b \bmod \varphi(n)}$.
In the same way a multiplication mod n is equivalent to an addition mod $\varphi(n)$ of the exponents, the modular exponentiation mod n is equivalent to a multiplication mod $\varphi(n)$ of the exponents.

Theorem(*Euler*) For all a coprime with n, it holds that:

$$a^{\varphi(n)} \equiv 1 \mod n \tag{2}$$

Multiplicative group of integers modulo n

So far, we have worked with $\mathbb{Z}_n$ with either addition or multiplication. Let us remember that a group requires four properties:

- closure
- associativity
- $\exists$ neutral (identity) element
- all elements of the group have an inverse

Working with the multiplicative group $\mathbb{Z}_8^*$ for instance, we would find that **not** all values in $\mathbb{Z}_8$ have an inverse, as shown in the below table.

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

We deduce from this table that the elements of $\mathbb{Z}_8^*$ are $\{1, 3, 5, 7\}$ because they have an inverse. More generally, any value a in $\mathbb{Z}_n$ coprime to n is in $\mathbb{Z}_n^*$.

Group order and element order

The order of a group refers to the cardinality of the group, i.e. the number of elements. The order of an element a is the smallest positive integer m such that $a^m = n$ where n is the neutral (or identity) element.

Exercises

Exercise 1

Compute as fast as possible, without writing 78130*8012*700451*19119 mod 20.

Answer of exercise 1

Working modulo 20, we can ignore multiples of 100 and hence keep only the two last digits of each numbers. We see that $78130 \equiv 30 \equiv 10$ and $8012 \equiv 12$. Since $12 \cdot 10$ is an obvious multiple of 20, the whole product is 0.

Exercise 2

Compute by exhaustive search 23^{-1} in $\mathbb{Z}_{57}$(the answer is a single digit number). Using this result, solve $23x + 52 \equiv 5$ in $\mathbb{Z}_{57}$. Could you solve an equation of the form $19x + a \equiv b$ using the same method?

Answer of exercise 2

- $23 \cdot 5 = 115 \equiv 1 \mod 57$.
- $x \equiv (5 - 52) \cdot 23^{-1} \equiv 50$
- No because 19 is not invertible as $57 = 19 \cdot 3$ (not coprime)

Exercise 3

Show that $n - 1$ is self inverse in $\mathbb{Z}_n$.

Answer of exercise 3

$(n-1)^2 = n^2 - 2n + 1 \equiv 1 \mod n$

Exercise 4

Show that for $n = pq$, $\varphi(n) = (p-1)(q-1)$ for p, q two prime numbers.

Answer of exercise 4

Let S_1 be the multiples of p less than pq and let S_2 be the multiples of q less than pq. Total number of coprimes $\varphi(pq) = pq - 1 - |S_1| - |S_2|$ since only multiples of p or q can divide pq. Since $|S_1| = q - 1$ and $|S_2| = p - 1$, we have $\varphi(pq) = pq - 1 - q + 1 - p + 1 = pq - p - q + 1 = (p - 1) \cdot (q - 1)$

Exercise 5

Compute 2^i mod 25 until cycling back to 1(it might take a while but less than 25 steps). Then:

- Deduce the value of $\varphi(25)$.
- Compute $18 * 22$ mod 25 without doing any multiplication using the previous results.
- Solve $16x \equiv 1$ mod 25.
- Compute 17^{2024} mod 25.

Answer of exercise 5

```
0 -> 1      11 -> 23
1 -> 2      12 -> 21
2 -> 4      13 -> 17
3 -> 8      14 -> 9
4 -> 16     15 -> 18
5 -> 7      16 -> 11
6 -> 14     17 -> 22
7 -> 3      18 -> 19
8 -> 6      19 -> 13
9 -> 12     20 -> 1
10 -> 24
```

- $\varphi(25) = 20$
- $18 \cdot 22 = 2^{15} \cdot 2^{17} = 2^{32} \equiv 2^{12} \equiv 21$ (remember we compute the exponent mod $\varphi(25) = 20$)
- $x \equiv 16^{-1}$
  $\Leftrightarrow x \equiv 2^{4^{-1}} \equiv 2^{-4}$
  $\Leftrightarrow x \equiv 2^{-4} \cdot 1 \equiv 2^{-4} \cdot 2^{20} \equiv 2^{16} \equiv 11$

- $17^{2024} \equiv 17^4 \equiv 2^{13 \cdot 4} \equiv 2^{52} \equiv 2^{12} \equiv 21$

Ex. 6 — Asymmetric Cryptography - Euler $\varphi(n)$ Function

1. Compute the Euler $\varphi(n)$ function for all $n \in \{2, 3, 4, 5, 36\}$.
2. Give the results of $2^{32} \bmod 31$, $3^{16} \bmod 32$ and $8^{14} \bmod 25$ without performing the actual exponentiations but by using only the Euler Theorem.

Answer of exercise 6

1.
   - $\varphi(2) = 2^1 - 2^0 = 2 - 1 = 1$
   - $\varphi(3) = 3^1 - 3^0 = 3 - 1 = 2$
   - $\varphi(4) = \varphi(2^2) = 2^1 - 2^1 = 4 - 2 = 2$
   - $\varphi(5) = 5^1 - 5^0 = 5 - 1 = 4$
   - $\varphi(36) = \varphi(2^2 3^2) = \varphi(2^2) \cdot \varphi(3^2) = (2^2 - 2^1) \cdot (3^2 - 3^1) = 2 \cdot 6 = 12$

2.
   - According to Euler Theorem we have $2^{30} = 2^{\varphi(31)} = 1 \bmod 31$.
     Therefore, we can compute $2^{32} \bmod 31 = 2^2 \cdot 2^{30} \bmod 31 = 4 \cdot 1 \bmod 31 = 4 \mod 31$.
     We conclude that $2^{30} \equiv 4 \pmod{31}$.
   - Similarly, according to Euler Theorem we have $3^{16} = 3^{\varphi(2^5)} = 3^{\varphi(32)} = 1 \bmod 32$.
     Therefore, $3^{16} \equiv 1 \pmod{32}$.
   - Since 8 and 25 are coprime, we can apply Euler's theorem. Let us first compute $\varphi(25)$. $\varphi(25) = \varphi(5^2) = 5^2 - 5^1 = 20$
     Because the exponent is lower than $\varphi(25)$, it is difficult to actually compute anything. However, we can still lower the exponent base to increase the exponent to a value greater than $\varphi(25)$: $8^{14} = (2^3)^{14} = 2^{42}$.
     We can now apply Euler's theorem: $2^{42} = 2^{20} \cdot 2^{20} \cdot 2^2 \equiv 1 \cdot 1 \cdot 2^2 \bmod 25 \equiv 4 \mod 25$.

Ex. 7 — Cyclic Groups and Generators

Working with the multiplicative group $\mathbb{Z}_p^*$ for $p = 19$ ...

1. List all the elements of $\mathbb{Z}_{19}^*$ and determine the order of the group.
2. Determine the order $\texttt{ord}(a)$ of each element $a \in \mathbb{Z}_{19}^*$. Use the following two facts to simplify the amount of calculations:

   Fact (1) If $a \in \mathbb{Z}_p^*$ then $\texttt{ord}(a)$ divides the order of $\mathbb{Z}_p^*$.

   Fact (2) $\texttt{ord}(a^k)$ is equal to $\texttt{ord}(a)/\texttt{gcd}(\texttt{ord}(a), k)$.

3. List all the generators of $\mathbb{Z}_{19}^*$.

Answer of exercise 7

1. Since p is prime, the order of the group $\mathbb{Z}_p^* = p - 1 = 19 - 1 = 18$. The elements of $|\mathbb{Z}_{19}^*|$ are $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18\}$.

2. Recall that the order of an element $a \in \mathbb{Z}_p^*$ is the smallest number i such that $a^i \mod p = 1$ where $1 \leq i \leq |\mathbb{Z}_p^*|$.

   Obviously, the order $\mathtt{ord}(1) = 1$.

   For any other value $a \neq 1$, we need to explore a wider range of possibilities. From Fact (1), we know that i divides $\mathtt{ord}(\mathbb{Z}_{19}^*) = 18$. As a result, the candidates for i are $\{1, 2, 3, 6, 9, 18\}$.

   Using Fact (2) we know that computing $\mathtt{ord}(2)$ will enable us to easily calculate $\mathtt{ord}(4)$, $\mathtt{ord}(8)$ and $\mathtt{ord}(16)$. Similarly, computing $\mathtt{ord}(3)$ will enable us to easily calculate $\mathtt{ord}(9)$.

   Finally, let us not forget that we from Euler's theorem, $a^{18} \equiv 1 \mod 19$ since $\varphi(19) = 18$.

   To sum up, what we need to do is to compute the order for the elements $a \in \{2, 3, 5, 6, 7, 10, 11, 12, 13, 14, 15, 17\}$ by finding the smallest integer $i \in \{2, 3, 6, 9\}$ such that

   $$a^i \mod 19 = 1.$$

   If such integer i doesn't exist then the order of a equals automatically to 18 (which is the order of the group $\mathbb{Z}_{19}^*$) from Euler's theorem.

   For 2:

   - $2^2 = 4$
   - $2^3 = 8$
   - $2^6 = 64 \equiv 7 \mod 19$
   - $2^9 = 2^3 \cdot 2^6 = 8 \cdot 7 = 56 \equiv 18 \mod 19$
   - Since none of the values worked, we deduce from Euler's theorem that $2^{18} \equiv 1 \mod 19$ and that $\mathtt{ord(2)} = 18$.

   This enables us to compute 4, 8 and 16 easily:

   - $4 = 2^2 \Leftrightarrow 2^{18} = (2^2)^9 \Rightarrow \mathtt{ord}(4) = 9$
   - $8 = 2^3 \Leftrightarrow 2^{18} = (2^3)^6 \Rightarrow \mathtt{ord}(8) = 6$

- $16 = 2^4$. From Fact (2) we know that $\mathtt{ord}(2^4) = \frac{18}{\mathtt{gcd}(\mathtt{ord}(2),4)} = \frac{18}{\mathtt{gcd}(18,4)} = \frac{18}{2} = 9$:

The complete list of $\mathtt{ord}(a)$ can be found in the below table.

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\mathtt{ord}(a)$	1	18	18	9	9	9	3	6	9	18	3	6	18	18	18	9	9	2

3. Since $\mathbb{Z}_{19}^*$ is a cyclic group (because 19 is a prime) the number of generators can be determined by computing $|\mathbb{Z}_{\varphi(p)}^*|$. Hence we need to calculate $|\mathbb{Z}_{\varphi(19)}^*| = |\mathbb{Z}_{18}^*|$. Applying Euler phi function this results in $|\mathbb{Z}_{18}^*| = \varphi(18) = 6$.

Université Libre de Bruxelles
Faculté des Sciences

2022-2023

INFO-F-405: Introduction to cryptography

4. Public-key techniques

Going public

Exercise 1

The characters in the text below make use of public-key cryptography. They each prepared a pair of public/private keys and they all know everyone else's public key. Complete the following sentences and scratch the invalid alternatives (as much as possible without looking at the slides or at your notes).

To send a confidential file to Xavier, Yasmina _____ it with her/Xavier's _____ key. When she receives it, she _____ it with his/her _____ key.

Elena sends to her husband Fabrice the list of items to buy at the grocery store. The list is not confidential, but she wants to avoid their daughter Gabi from adding chocolates or other extra items to it. To do so, she _____ the list with her/Fabrice's/Gabi's _____ key before she sends it. When he receives it, Fabrice _____ the _____ with his/Elena's/Gabi's _____ key.

Answer of exercise 1

To send a confidential file to Xavier, Yasmina **encrypts** it with **Xavier's public** key. When she receives it, she **decrypts** it with **her private** key.

Elena sends to her husband Fabrice the list of items to buy at the grocery store. The list is not confidential, but she wants to avoid their daughter Gabi from adding chocolates or other extra items to it. To do so, she **signs** the list with **her private** key before she sends it. When he receives it, Fabrice **verifies** the **signature** with **Elena's public** key.

Exercise 2

Thelma and Louise recently met for the first time and would like to do business together. After their meeting, because of the COVID-19, they are forced to stay at

home and can communicate only by phone or via the Internet. They wish to start discussing their business project, but want to do so confidentially. Hopefully, they already exchanged their phone numbers and email addresses, and are both knowledgeable about cryptography. However, their project is highly sensitive and they need to protect their conversation against even active adversaries, that is, people who can afford to tamper with Internet connections. We nevertheless assume that the adversaries are not going to cut their communication lines, as this would be too obvious and as they would otherwise miss all opportunities of spying on the two business women.

Please explain how Thelma and Louise can set up a secure communication channel using public-key cryptography.

Answer of exercise 2

Let us first identify their problems:

- They want to communicate confidentially.
- They cannot meet in person.
- Someone could intercept and alter their internet communication.
- Someone could intercept their phone communication.

Since Thelma and Louise cannot meet and any of their communications can be heard, they cannot exchange any secret key directly. Instead, they could use public key cryptography, but Thelma and Louise need to exchange their public keys while avoiding man-in-the-middle (MITM) attacks. To be able to encrypt with the other party's public key, they must be sure that the public key that they received indeed belongs to the right person. To ensure this, Thelma and Louise can simply call each other and verify each other's public key fingerprint (i.e., the hash of the public key). We assume that they are able to recognize the each other's voices and that an adversary cannot simulate their voices while they speak out their public key fingerprints.

To exchange the public keys, one could say that Thelma and Louise can rely on a public key infrastructure (PKI). However, it is not so clear how to use this properly. The goal would be that Thelma and Louise obtain a certificate on their public keys, but how to avoid a MITM on that part? This only moves the problem elsewhere: How can the certification authority (CA) ensure that the public key indeed belongs to Thelma or to Louise?

Exercise 3

Alice has recently arrived in Belgium for a one-year Erasmus at ULB/VUB. Just like any other student, she has been granted access to the university supercomputer, *Hydra*, to run experiments for her Master's Thesis.

We will assume that the connection protocol with Hydra consists in a Diffie-Hellman key exchange followed by the symmetric encryption of the communications. Upon the first connection, Alice's computer requests Hydra's long-term public key and stores it for the subsequent connections. Then for each connection, Alice's computer generates an ephemeral key pair and sends the public key to Hydra. Finally, each party generates a secret key from its private key and the public key received from the other party.

Carelessly, Alice connects to Hydra with this protocol from her student residence.

a. Let us assume that an attacker (Charles) has control over the network of Alice's student residence. Charles knows that many students will try to connect to Hydra. How can Charles intercept and read all communications between Alice's computer and Hydra in the clear without being noticed?

b. The next day, Charles does not intercept connections to Hydra anymore. What will Alice notice when she connects to Hydra? Why?

c. In SSH, the first time you connect to a host, you get prompted with a *fingerprint* and asked whether that *fingerprint* matches what you expect from the remote host you are trying to reach. What is this fingerprint? To what is it applied? What kind of attack does it prevent?

d. Let us assume that there is a trusted authority such as the Belgian government in the loop. Could the ULB/VUB prevent attacks such as the one Charles is able to perform via this trusted authority ?

Answer of exercise 3

a. Since Charles has control over Alice's network, he can redirect the traffic going to Hydra to his own machine and thereby perform a man-in-the-middle (MITM) attack.

   Consequently, when Alice will connect to Hydra and ask its public key, Charles will send his own instead, and forward all Alice's requests to the actual Hydra. Since Alice does not know the public key of Hydra, she will not notice anything.

b. Because Charles is not decrypting/re-encrypting anymore, Alice will notice that her messages are not understood by Hydra and vice-versa.

   To see this, remember that Charles' public key was stored on Alice's computer as presumed Hydra's public key. Hence, Alice generates a secret key using her ephemeral key pair and Charles' public key. The Hydra server, on its side, generates a secret key using Alice's ephemeral public key and its actual private key (and not Charles'). The generated secret keys therefore do not match between Alice and Hydra, and the traffic encrypted to the other party will decrypt as garbage.

c. This fingerprint is a hash of the public key. If the user knows the remote host fingerprint, (s)he can cross-check it at first connection. This mechanism prevents MITM attacks provided the fingerprint is securely communicated beforehand.

d. Introducing a trusted authority in the loop enables us to rely on a Public Key Infrastructure (PKI). In a PKI, a trusted authority can provide certificates stating that a given public key indeed belongs to a given user (domain name). As a result, provided Hydra has obtained a certificate signed by the Belgian Government, Alice can request that certificate when connecting and thereby check the identity of the remote host.

   This reasoning only holds assuming that Alice knows the public key of the Belgian government, which is a fair assumption to make.

   However, if we assume that this assumption does not hold, the trusted authority does not solve the problem since Alice would have to retrieve the public key of the Belgian government, which could also end up in a MITM attack...

Exercise 4

Lets say (Gen, E, D) is a semantically secure public-key encryption system. Could the algorithm E be deterministic?

Answer of exercise 4

No, because we want two ciphertexts of the same plaintext to be different.

Rivest-Shamir-Adleman

Exercise 5

RSA Encryption

a. Consider the key generation algorithm of RSA. Let $p = 11$ and $q = 17$ be two given prime numbers. What is the corresponding RSA modulus n? Assuming that the public exponent is $e = 3$, what is the corresponding private exponent d in the private key?

b. Consider the encryption algorithm of the RSA "textbook encryption" scheme. Assuming that the plaintext is $m = 15$ and the public key (n, e) is as in question a, compute the resulting ciphertext c.

c. Consider the decryption algorithm of the RSA "textbook encryption" scheme. Using a computer, check that the decryption of the ciphertext c from the previous question indeed yields the original plaintext $m = 15$.

d. Consider the figure depicting RSA-OAEP in the slides and assume a 16-bit RSA modulus (hence $n = 16$ in the notation of that figure), 8 bits of randomness (hence $k0 = 8$) and 4 bits of redundancy (hence $k1 = 4$).

   The functions $G, H : \{0,1\}^* \to \{0,1\}^*$ used in RSA-OAEP should be instantiated with extendable output functions (or hash functions with the MGF1 mode). For simplicity, however, we will assume in this question that they take a very simple form. With $\kappa = k0 = n - k0 = 8$, G and H are defined as follows:

   $$G : b_0 \dots b_{\kappa-1} \mapsto b_{\kappa-1} \dots b_0 \qquad \text{and} \qquad H : b_0 \dots b_{\kappa-1} \mapsto \overline{b_0} \dots \overline{b_{\kappa-1}},$$

   where $\overline{b_i} = 1 - b_i$ for all $0 \leq i < \kappa$.

   Let the plaintext be $m = 1010$ and the randomness be $r = 1110\,0101$. Compute the resulting input $(X \| Y)$ of the exponentiation and give its numerical value in decimal.

e. Let's say we want to set up an RSA system for k users. How many primes do we have to generate? What security issues we might create if we use less prime numbers?

Answer of exercise 5

a. The RSA modulus is $n = 11 \times 17 = 187$. The relationship between RSA exponents e and d is given by the following equation: $ed \bmod \phi(n) = 1$. In other words d is the (multiplicative) inverse of e modulo $\phi(n)$. We first compute $\phi(n) = \phi(187) = \phi(11)\phi(17) = 10 \times 16 = 160$. Note that since $\gcd(3, 160) = 1$ the existence of this inverse is guaranteed.

   We can compute the inverse of 3 mod 160 by computing the extended GCD algorithm, but this was not seen in this course. Alternatively, we can start

from Bézout's identity: There exist integers x and y such that $3x + 160y = \gcd(3, 160) = 1$, and x is the inverse we are looking for, as $(3x + 160y) \bmod 160 = 3x \bmod 160 = 1$. Since $160 = 3 \times 53 + 1$, we can rewrite it as $3(x + 53y) + y = 1$. Setting $x' = x + 53y$, we have $3x' + y = 1$, which can be solved by taking $x' = 1$ and $y = -2$. Finally, $x' = 1$ means that $x + 53 \times (-2) = 1$ and $x = 107$. We can check that indeed $(3 \times 107) \bmod 160 = 1$.

The corresponding private exponent is therefore $d = 107$.

b. Textbook RSA ciphertexts are computed as $c = m^e \bmod n$. Hence, $c = 15^3 \bmod 187 = 3375 \bmod 187 = 9$.

c. The decryption process of RSA "textbook encryption" requires us to compute $m = c^d \bmod n$. Hence we need to compute $m = 9^{107} \bmod 187 = 15$. One can use for instance Python: `(9 ** 107) % 187`.

d. First we extend m with $k1 = 4$ zeroes and get $m' = 1010\,0000$. Then, we compute $G(r) = 1010\,0111$ and get $X = G(r) \oplus m' = 0000\,0111$. Next, we compute $H(X) = 1111\,1000$ and get $Y = r \oplus H(X) = 0001\,1101$. The result is $X \| Y = 0000\,0111\,0001\,1101$ and this is interpreted as the integer 1821.

e. We need at least $2k$ primes, otherwise the users sharing a factor could easily recover the private key of each other.

Exercise 6

In the RSA "textbook signature" scheme, a message m is signed in the following way under the private key (n, d):

$$m \to (m, s) = (m, m^d \bmod n),$$

Is there any way of creating a forgery?

Answer of exercise 6

Yes, starting from a valid message-signature pair $(m, s) = (m, m^d)$, the adversary chooses any value c and constructs the following new message-signature pair:

$$(m', s') = (m \times c^e, s \times c).$$

To see that (m', s') is a forgery, let us verify the signature with the public key:

$$(s')^e = (s \times c)^e = s^e \times c^e = m^{ed} \times c^e = m \times c^e = m'.$$

In other words, (m', s') would be accepted as a valid message-signature pair, although it was not signed with the private key.

Exercise 7

Alice frequently needs to upload files to Bob's server, and they use RSA and hybrid encryption to keep their files confidential. Please find below the specifications of their protocol, similar to RSA-KEM, to which we added a few mistakes.

One-time setup

1. Bob sets $e = 3$.
2. Bob randomly chooses a private prime number p of 256 bits. He checks that $\gcd(e, p) = 1$; otherwise, he repeats with a new prime p.
3. Similarly, Bob randomly chooses another private prime number q of 256 bits. He checks that $\gcd(e, q) = 1$ and $p \neq q$; otherwise, he repeats with a new prime q.
4. Bob computes $d = e^{-1} \bmod \phi(pq)$ and keeps d private.
5. Bob computes $n = pq$ and sends (e, n) to Alice. They check together that Alice received Bob's public key correctly.

When Alice needs to upload a file F

6. Alice chooses a random string m of 512 bits.
7. Alice computes the 160 bits of output of $\mathrm{SHA1}(m)$ and interprets them as the integer k with $0 \leq k \leq 2^{160} - 1$.
8. Alice computes $c = k^e \bmod n$, and she encrypts her file F with a good symmetric encryption scheme using the secret key k resulting in ciphertext G.
9. Alice uploads (c, G).

When Bob receives an encrypted file (c, G)

10. Bob recovers k by computing $k = c^d \bmod n$.
11. Bob chooses a random string m of 512 bits and computes $k' = \mathrm{SHA1}(m)$. If $k' \neq k$, it outputs an error message and aborts.
12. Bob decrypts G with the same good symmetric encryption scheme, using k as secret key, to recover F.
13. Bob stores F on the server.

Questions

a. What is the size in bits of Bob's modulus n that will result from the setup? Does that give sufficient security in 2021? If so, please justify briefly. If not, please recommend which size the primes should have to achieve about 128 bits of security.

b. On line 4, what is $\phi(pq)$? Can you give a simpler expression? What would happen if $\phi(pq)$ was part of Bob's public key?

c. There is a mistake in the one-time setup procedure that can cause the computation to fail sometimes. What is it? How can you fix it, and why? Before it is fixed, what is approximately the probability that this procedure fails?

d. There is a mistake in the way RSA is used that causes a major security issue. What is it? How can you fix it, and why?

e. There is a silly mistake in Bob's procedure that causes it to fail almost always. What is it? How can you fix it, and why?

f. Can someone other than Alice upload a file onto Bob's server? If so, please sketch briefly how to modify the protocol so that only Alice can upload a file. Otherwise, please explain briefly how the current protocol achieve this restriction.

Answer of exercise 7

a. Because Bob constructs the modulus n as the product of two 256-bit integers, n will be 511 or 512-bit long. This does not give sufficient security nowadays, as a 512-bit integer was factored in 1999 and, at of the time of this writing, the record is the factorization of a 829-bit integer, see https://en.wikipedia.org/wiki/RSA_Factoring_Challenge. To meet 128-bit security, the NIST currently recommends a 3072-bit modulus.

b. Here $\phi(n) = \phi(pq) = (p-1)(q-1)$ since p and q are distinct primes.

If $\phi(n)$ was made public, one could easily find d from e since $d = e^{-1} \bmod \phi(n)$. And we saw (see slides) that the knowledge of e and d allows one to factor n easily, hence to break the entire scheme. Alternatively, from n and $\phi(n)$, one could deduce the sum of p and q: $n-\phi(n)+1 = pq-(p-1)(q-1)+1 = p+q$. Finding p and q from their sum and product comes down to solving a quadratic equation.

c. On lines 2 and 3, Bob checks that $\gcd(e, pq) = 1$; since e, p and q are distinct primes, this is always the case. However, he should check that $\gcd(e, (p-1)(q-1)) = 1$ instead; this check is required to make sure that e has a multiplicative inverse modulo $\phi(n)$ and hence that d exists.

   Before it is fixed, the computation fails when $\phi(n) = (p-1)(q-1)$ is a multiple of $e = 3$. We have $\Pr[3 \text{ does not divide } p-1] \approx \frac{2}{3}$, and similarly for q. So $\Pr[3 \text{ does not divide } (p-1)(q-1)] \approx \frac{4}{9}$ and the failure probability is about $1 - \frac{4}{9} = \frac{5}{9}$.

d. There is a short message attack!

   We have $k < 2^{160}$ and therefore $k^e = k^3 < 2^{480}$. But since n is at least 2^{510} (and more likely n is even much bigger as seen in question a), we have that $k^e < n$ and the modular reduction (on line 8) has no effect. An attacker can therefore easily recover k by computing $\sqrt[3]{c}$ in the plain integers.

   The best way to fix this is by following RSA-KEM more closely: 1) Alice chooses m of the same size as n, 2) Alice computes $c = m^e \bmod n$ (instead of k^e) and 3) Bob first recovers m and then gets k by hashing m.

e. Line 11 is pointless and is most likely going to fail. We can simply remove it.

f. Yes, assuming that Bob's public key (e, n) is indeed public, anyone can upload a file onto Bob's server. Bob does not check that the file comes from Alice. Furthermore, Bob has no way to check the authenticity of the file because he has no way to distinguish Alice from someone else.

   Alice could also create a pair of public and private keys, share her public key with him and check with him that he received it correctly (to avoid MITM attacks). This way, Alice could sign the file that she uploads and Bob could verify the signature before storing it onto the server.

Discrete logarithm problem in $\mathbb{Z}_p^*$

Exercise 8

The discrete logarithm problem is based on the multiplicative group $(\mathbb{Z}_p^*, \times)$. What happens if we would use the additive group $(\mathbb{Z}_p, +)$ instead? *Hint:* Start with writing out the problem in that group.

Answer of exercise 8

The discrete "logarithm" problem is easy to solve in the additive group.

To see this, let us fix a generator $g \in \mathbb{Z}_p$. Then, the problem is the following: Given $A = ag \bmod p$, find A. It is sufficient to compute $a = g^{-1}A \bmod p$ with $g^{-1} \bmod p$ the multiplicative inverse of g modulo p.

Exercise 9

ElGamal encryption

a. Let $\mathbb{G}$ be a subgroup of $\mathbb{Z}_p^*$, $p = 23$. The order of $\mathbb{G}$ is $|\mathbb{G}| = 11$. Let $g = 4$ be a generator of $\mathbb{G}$. Consider the key generation algorithm of the ElGamal encryption scheme (and of all DLP-based schemes). Assuming that the private key is $a = 3$, compute the corresponding public key A.

b. Consider the encryption algorithm of the ElGamal encryption scheme. Assume that the plaintext $m = 3$, the random exponent (ephemeral private key) chosen by the encryption algorithm is $k = 2$, and $\mathbb{G}$, g, and A are as in question a. Compute the resulting ciphertext (K, c).

c. Consider the decryption algorithm of the ElGamal encryption scheme. Assume that the ciphertext is $(K, c) = (6, 22)$ and $\mathbb{G}$, g, a, and A are as in question a. Compute the resulting plaintext m.

Answer of exercise 9

a. We have that the public key $A = g^a$ in $\mathbb{G}$. Hence $A = g^a \bmod 23 = 4^3 \bmod 23 = 64 \bmod 23 = 18$.

b. ElGamal ciphertexts (K, c) are computed as follows: $K = g^k$ and $c = A^k m$ in $\mathbb{G}$. We thus compute $K = 4^2 \bmod 23 = 16$ and $c = 18^2 \cdot 3 \bmod 23 = 324 \cdot 3 \bmod 23 = 6$. The ciphertext is thus $(K, c) = (16, 6)$.

c. The decryption process of ElGamal encryption requires us to first compute K^a and its inverse in $\mathbb{G}$. We first compute $K^a = 6^3 \bmod 23 = 9$. Now we need to compute the inverse of 9 mod 23 by using the extended euclidian algorithm and find it is equal to 18, so $K^{-a} = 18$ in $\mathbb{G}$. The next step in the decryption process is the computation of $m = c \cdot K^{-a}$ in $\mathbb{G}$. Hence we can compute $m = 22 \cdot 18 \bmod 23 = 5$.

Exercise 10

What is the additive group notation for $\mathbb{Z}_p^*$? Given a prime number p, let the group $\mathbb{G}$ be the set of integers between 1 and $p-1$ (like $\mathbb{Z}_p^*$) equipped with the group operation $+$ be the multiplication modulo p. (Yes, an "addition" symbol to denote a multiplication in this case!) The neutral element of $\mathbb{G}$ is denoted O, i.e., $O = 1$, and the inverse of an element A is denoted $-A$.

The *scalar multiplication* refers to the process of repeating the group operation onto itself a given number of times. We denote $[n]A$ the group element obtained by repeating the group operation on n copies of A. So, $[0]A = O$, $[1]A = A$, $[2]A = A + A$, $[3]A = A + A + A$, etc, and $[-1]A = -A$, $[-2]A = (-A) + (-A)$, etc.

For instance, let $p = 23$ as above, let A and B be group elements $A = 4$ and $B = 6$. Then $A + B = O$ as $4 \times 6 \equiv 1 \pmod{23}$. So A and B are each other's inverse, i.e., $A = -B$. Also, $[5]A = 12$ since $4^5 \equiv 1024 \equiv 12 \pmod{23}$. And $[-1]A = -A = 6$.

ElGamal encryption in additive group notation Assuming a group $\mathbb{G}$, a generator $G \in \mathbb{G}$, rewrite the key generation process to generate Alice's public-private key pair in additive group notation. Then, given a plaintext message $M \in \mathbb{G}$, rewrite the ElGamal encryption scheme in the same notation. Finally, rewrite the decryption and explain why it correctly recovers the plaintext.

Answer of exercise 10

Alice generates a public-private key pair as follows:

- Privately choose a random integer in $a \in \mathbb{Z}_q \setminus \{0\}$, with $q = |\mathbb{G}|$
- Compute $A = [a]G$

The public key is A and the private key is a.

To encrypt $M \in \mathbb{G}$ with Alice's public key A:

- Choose randomly an integer $k \in \mathbb{Z}_q \setminus \{0\}$
- Compute
$$\begin{cases} K = [k]G \\ C = M + [k]A \end{cases}$$

To decrypt, Alice computes:
$$M = C + [-a]K$$

Why is it correct?

$$[-a]K = [-ak]G = [-k]A = (-[k]A)$$
$$C + [-a]K = M + [k]A + [-a]K = M + [k]A + (-[k]A) = M$$

Exercise 11

Let's say Alice and Bob both choose a planet that they want to visit (in our solar system). They want to check if they choose the same planet without giving out their choice. Let's say Alice choses the planet a and Bob the planet b. Alice and Bob agree on the following scheme:

- They publicly choose a prime p and generator g of $G = \mathbb{Z}_q^*$
- Alice chooses random x and y in $\mathbb{Z}_p$ and sends to Bob $(A_0, A_1, A_2) = (g^x, g^y, g^{xy+a})$
- Bob choose random r and s in $\mathbb{Z}_p$ and sends back to Alice $(B_1, B_2) = (A_1^r \times g^s, \left(\frac{A_2}{g^b}\right)^r \times A_0^s)$

How Alice can check if she had chose the same planet as Bob?

Answer of exercise 11

Check whether B_1^x is equal to B_2.

Exercise 12

Schnorr Signature

a. In contrast to RSA with full-domain hashing, the Schnorr signature scheme is probabilistic, i.e., different executions of the signing algorithm on the same input (a, m) results in different signatures $\sigma = (s, e)$. How many different Schnorr signatures can exist for a single message $m \in \{0,1\}^*$?

b. Let q the size of the group, i.e., $q = p - 1$ if g is a generator of $\mathbb{Z}_p^*$. Assume that during the generation of some Schnorr signature $\sigma = (s, e)$ the random exponent (or ephemeral private key) $k \in \mathbb{Z}_q$ becomes known to an adversary $\mathcal{A}$. How can $\mathcal{A}$ use σ and k in order to forge a Schnorr signature σ' for any message m' of its choice?

c. Assume that the same random exponent (or ephemeral private key) $k \in \mathbb{Z}_q$ used in the signing algorithm of the Schnorr scheme was used multiple times. More precisely, we have two signatures $\sigma_1 = (s_1, e_1)$ and $\sigma_2 = (s_2, e_2)$ on two different messages $m_1 \neq m_2$ that were generated using the same exponent $k_1 = k_2 = k$. How can an adversary $\mathcal{A}$ use σ_1 and σ_2 in order to forge a Schnorr signature σ' for any message m' of its choice?

d. To make k random and unique per signature, the signer generates a random $k \in \mathbb{Z}_q$ upon the first signature, and then increments it ($k \leftarrow k + 1$) for each new signature. Is that secure? If so, please justify. If not, what is the problem and how can you fix it?

e. Propose two ways to pick k in a secure way. *Hint:* One probabilistic and one deterministic.

Answer of exercise 12

a. Take a closer look at the signing algorithm. For each choice of $k \in \mathbb{Z}_q$ value, $r = g^k$ and $s = k - ea$ are uniquely determined by m. Since the mapping $k \mapsto g^k$ is a bijection, i.e., for every $k \in \mathbb{Z}_q$ there exists exactly one group element $r = g^k$ and for every group element g^k there exists exactly one $k \in \mathbb{Z}_q$, we can conclude that there exist exactly $|\mathbb{Z}_q| = q$ different Schnorr signatures $\sigma = (s, e)$ for each message m.

b. $\mathcal{A}$ knows that $s = k - ea \bmod q$ and $e = \text{hash}(r \| m)$. From this $\mathcal{A}$ can compute the private key $a = e^{-1}(k - s) \bmod q$ and start forging signatures for arbitrary m' using the original signing algorithm of the Schnorr scheme.

c. $\mathcal{A}$ knows that $s_1 = k_1 - e_1 a \bmod q$ and $s_2 = k_2 - e_2 a \bmod q$, where hash values e_1 and e_2 are also known to $\mathcal{A}$. Since $k_1 = k_2 = k$, we subtract the two equations and we get the equality $s_1 - s_2 = a(e_2 - e_1) \bmod q$, which can be rewritten as $a = (e_2 - e_1)^{-1}(s_1 - s_2) \bmod q$. Note that $e_1 = e_2$ may occur only with a negligible probability since the hash function used to compute them is assumed to be collision-resistant. Hence, $\mathcal{A}$ can use $\sigma_1 = (s_1, e_1)$ and $\sigma_2 = (s_2, e_2)$ to compute the private key a and start forging signatures for arbitrary m' using the original signing algorithm of the Schnorr scheme.

d. Let us assume that a first message was signed using k followed immediately by a second message signed using $k + 1$. So, $\mathcal{A}$ knows that $s_1 = k - e_1 a \bmod q$ and $s_2 = (k + 1) - e_2 a \bmod q$. Again, we subtract the two equations and we get the equality $s_1 - s_2 + 1 = a(e_2 - e_1) \bmod q$, which can be rewritten as $a = (e_2 - e_1)^{-1}(s_1 - s_2 + 1) \bmod q$. Like in the previous question, the adversary can recover the private key a and start forging signatures for arbitrary messages.

To conclude, not only the ephemeral key k must be secret and unique, it must also be chosen independently upon each signature.

e. We can choose k randomly and uniformly in its range using a physical random bit generator, independently upon each signature, or choose k deterministically by hashing together m and a secret value, e.g., $k = \text{hash}(a \| m)$, to ensure only one possible signature per message.

Elliptic curve cryptography

Exercise 13

Elliptic curve in $\text{GF}(11)$**.** In this exercise, all the arithmetic operations are understood modulo 11, although not explicitly written. Let E be the curve over $(\text{GF}(11))^2$ satisfying the Weierstrass equation

$$y^2 = x^3 - 3x + 7.$$

a) List the points on the curve. To do this, we advise the following steps:

   - Build a table of squares modulo 11. For each value y, compute y^2. Sort the table per value y^2. How many squares modulo 11 are there? For a given value y^2, how many corresponding values y can there be? When more than 1, how do these values relate to each other?
   - For each value x, compute $x^3 - 3x + 7$. Using the table of squares modulo 11, look up the possible value(s) of y (if any) that satisfy $y^2 = x^3 - 3x + 7$.

b) How many points does E have? What is the largest prime-order group we can get? What is the cofactor?

c) Which point has order 1? Which point has order 2? (Hint: think about the elliptic curves over the reals.)

Answer of exercise 13

a) - There are six squares modulo 11. These are 0, 1, 3, 4, 5 and 9. The value 0 has one corresponding square root (itself), while the other squares have two possible square roots (a and b) that add up to 11, i.e., one is the opposite of the other ($a + b = 0 \mod 11$). The other values (i.e., the non-squares) have no square roots in $\text{GF}(11)$.

x	$x^2 \mod 11$
0	0
1	1
2	4
3	9
4	5
5	3
6	3
7	5
8	9
9	4
10	1

- The points on E are $(1, \pm 4)$, $(2, \pm 3)$, $(3, \pm 5)$, $(4, \pm 2)$, $(8, 0)$, $(9, \pm 4)$, $(10, \pm 3)$ plus O the point at infinity.

x	$y^2 = x^3 - 3x + 7$	y
0	7	
1	5	4, -4
2	9	3, -3
3	3	5, -5
4	4	2, -2
5	7	
6	7	
7	10	
8	0	0
9	5	4, -4
10	9	3, -3

b)
- There are thus 14 points.
- The largest prime-order group has size 7.
- The co-factor is 2.

c)
- The point at infinity O has order 1 because it is the neutral element in the group.
- Point $(8, 0)$ has order 2. In the reals, the point with $x = 0$ has a vertical tangeant, and this property carries over to curves over finite fields, hence $[2](8, 0) = O$.

Exercise 14

ElGamal signature with elliptic curves The purpose of this exercise is to translate the ElGamal signature scheme as expressed with modular exponentiation into an elliptic curve-based scheme.

First, we recall the original scheme here. Given a prime p and a generator g over $\mathbb{Z}_p^*$, we proceed as follows.

- **Signature** of message $m \in \mathbb{Z}_2^*$ by Alice with her private key a:
  - Compute $h = \text{hash}(m)$
  - Choose randomly an integer $k \in [1, p-2]$
  - Compute $r = g^k \bmod p$
  - Compute $s = k^{-1}(h - ar) \bmod (p-1)$
    - If $s = 0$, restart with a new k
  - Send (r, s) along with m

- **Verification** of signature (r, s) on m with Alice's public key $A = g^a \bmod p$:
  - Compute $h = \text{hash}(m)$
  - Check $A^r r^s \stackrel{?}{\equiv} g^h \pmod p$

Now, let E be an elliptic curve over $\text{GF}(p)$ and let $G \in E$ be a generator of prime order q. Rewrite the scheme above by replacing operations in $\mathbb{Z}_p^*$ with operations in E. (Hint: to compute the equivalent of r, first compute $R = [k]G$ then let r be the x-coordinate of R.)

What is the equivalent of operations modulo $p - 1$?

Answer of exercise 14

Given an elliptic curve E and a generator $G \in E$ of prime order q, we proceed as follows.

- **Signature** of message $m \in \mathbb{Z}_2^*$ by Alice with her private key $a \in \mathbb{Z}_q$:
  - Compute $h = \text{hash}(m)$
  - Choose randomly an integer $k \in [1, q-1]$
  - Compute $R = [k]G$ then let r be the x-coordinate of R
  - Compute $s = k^{-1}(h - ar) \bmod q$

    - If $s = 0$, restart with a new k
  - Send (R, s) along with m

- **Verification** of signature (R, s) on m with Alice's public key $A = [a]G$:
  - Compute $h = \text{hash}(m)$
  - Let r be the x-coordinate of R
  - Check $[r]A + [s]R \stackrel{?}{\equiv} [h]G$

The equivalent of operations modulo $p - 1$ are operations modulo q, in both case modulo the order of the generator.

Exercise 15

A company that installs and maintains an open-source operating system is creating an automatic update mechanism for its customers. Regularly, the company publishes an *update pack* containing the latest changes to be made to the operating system, and each computer connected to the Internet can fetch it. The company would like to guarantee the integrity of these update packs so as to avoid the installation of malicious software on their customers' computers. Fortunately, confidentiality is not required, as it is open-source software.

In the following, we describe a protocol that the company uses to sign the update packs and for the customers to check that the update packs are genuine. However, we voluntarily added some mistakes (including omissions) to its definition. Some of them are functional, that is, they prevent the protocol from working correctly, while others introduce security flaws.

Please *list all the mistakes* you find, and for each, *justify* why it is incorrect and *propose a correction*.

Let $\mathcal{E}$ be a standard elliptic curve over $\text{GF}(p)$ (for some prime p) that is used by the company and its customers. They also agreed on a base point $G \in \mathcal{E}$ with prime order q, i.e., $[q]G$ is the neutral element. Note that the uppercase letters refer to points on $\mathcal{E}$, while lowercase letters are integers, and `UP` is a string of bits that represents an update pack.

One-time setup at the company:

1. The company secretly generates its secret key c randomly and uniformly in $[1 \dots p - 1]$.

2. The company computes its public key $C = [c]G$, and publishes it via some PKI process. (This aspect is out of scope of this question. In the sequel, we assume that all the public keys are correctly authenticated and can be trusted.)

3. The company secretly generates its secret value n_C randomly and uniformly in $[1 \dots p-1]$.

Company's procedure to sign and post an update pack *UP*

1. Compute $k = \text{hash}(n_C)$.
2. Compute $R = [k]G$.
3. Compute $e = \text{hash}(R \| \mathit{UP})$.
4. Compute $s = k + ec \bmod p$.
5. Post (UP, e, s) on the company's update repository.

Customer's procedure to retrieve and install an update pack

1. Retrieve (UP, e, s) from the company's update repository.
2. Compute $C = [c]G$.
3. Compute $R' = [s]G - [e]C$.
4. Compute $e' = \text{hash}(R' \| \mathit{UP})$.
5. The customer installs *UP*.

Answer of exercise 15

The protocol is essentially a Schnorr signature on elliptic curves.

- A first problem is on line 1 of the setup procedure. The private key must be chosen in the set $[1 \dots q-1]$, with q the size of the group that G generates. The prime p does not play a role as a scalar here, it only plays a role in the coordinates (x, y) of the points on the curve, but this is abstracted away in the notation.

- A second problem is on line 1 of the company's procedure. The value k is computed deterministically from the fixed quantity n_C, hence k is fixed. This is a major security problem, as the signature of two different messages with the same value k allows anyone to recover the private key c.

  There are two possible ways to fix this:

- Draw k randomly in the set $[1 \dots q-1]$. (In this case, line 3 of the setup procedure can be discarded.)
- Derive k from the message and a secret value, e.g., $k = \text{hash}(n_C \| \mathsf{UP})$ similarly to EdDSA. (In this case, n_C does not need to be a number but rather a secret string of bits.)

- A third problem is on line 4 of the company's procedure. The computation must be done modulo q and not p, for the same reasons as above. (Some pointed out that the $+$ should be a $-$, as in the original Schnorr signature, but this expression is consistent with the way the verification is done.)

- A fourth problem is on line 2 of the customer's procedure. The customer does not have access to the private key of the company! Anyway, this step is useless because C is public and assumed to be trusted.

- A fifth and last problem is on line 5 of the customer's procedure. The protocol misses the verification that $e = e'$. The customer must check this before installing the update pack, and abort if $e \neq e'$.

Exercise 16

Projective coordinates. Instead of representing points on the curve with (x, y), called *affine coordinates*, one can use an alternate representation using three coordinates $(X : Y : Z)$ called *projective coordinates*.

When $Z \neq 0$, a point in projective coordinates $(X : Y : Z)$ represents (x, y) in affine coordinates with $x = XZ^{-1}$ and $y = YZ^{-1}$. The projective coordinates are *redundant*: For any $\lambda \neq 0$, the projective coordinates $(X : Y : Z)$ and $(\lambda X : \lambda Y : \lambda Z)$ represent the same point.

In projective coordinates, the Weierstrass equation becomes

$$Y^2Z = X^3 + aXZ^2 + bZ^3 \tag{1}$$

a) How can we represent the point at infinity O in projective coordinates? Check that it satisfies the Weierstrass equation above. (Hint: intuitively, the point at infinity is the vertical direction. Over the reals, it can be viewed as having affine coordinates $(0, \pm\infty)$. In the projective plane, points at infinity have $z = 0$ by definition.)

b) In affine coordinates, the addition of two points $P + Q$ in the general case, i.e., assuming $P \neq Q \neq -P$ and $P \neq O \neq Q$, works as follows. We have

$(x_P, y_P) + (x_Q, y_Q) = (x, y)$ with

$$
\begin{aligned}
s &= \frac{y_P - y_Q}{x_P - x_Q} \\
x &= s^2 - x_P - x_Q \\
y &= s(x_P - x) - y_P
\end{aligned}
$$

Given the projective coordinates of P and Q, say, $(X_P : Y_P : Z_P)$ and $(X_Q : Y_Q : Z_Q)$, write the result of the addition in projective coordinates $(X : Y : Z)$. As the projective coordinates are redundant, the value Z can be freely chosen; you may for instance fix $Z = 1$.

c) The projective coordinates are interesting in practical implementations for their higher efficiency: The operations on the points can be expressed *without inversions*, which are otherwise costly compared to additions, subtractions and multiplications. Rewrite the addition of points above, but using only additions, subtractions and multiplications (no inversions). (Hint: set Z so as to cancel any inversions.)

Answer of exercise 16

a) The point at infinity is defined as the point with $Z = 0$. Thus, we can find the values of X and Y that satisfy the Weierstrass equation. We find that the point at infinity is represented as $(0 : Y : 0)$ with $Y \neq 0$ (otherwise it would be the origin), since Y can take any value to fit the equation. We can check that this point indeed satisfies equation 1.

b) Set $Z = 1$ and just rewrite the equations above with $x_P = X_P Z_P^{-1}$ and $y_P = Y_P Z_P^{-1}$.

c) Set $Z = Z_P Z_Q (X_Q Z_P - X_P Z_Q)^3$, which cannot be zero because of the assumptions on P and Q.