Université Libre de Bruxelles
Faculté des Sciences

January 25, 2023

INFO-F-405: Introduction to cryptography

This assessment may be done with the use of written or printed personal notes, slides, books, etc., but without any electronic devices (e.g., laptop, tablet, phone) and without internet access. Also, the assessment is strictly personal and you are not allowed to communicate with other students or other people during this assessment.

This exam is made of 11 questions and includes a portfolio at the end. Please ensure that we can split your answers to questions 1-7 from those of questions 8-11 by using separate sheets of paper.

Principles

■ **Question 1** (10%): With your own words, please explain what are confidentiality and authenticity in cryptography. Then, does an encryption scheme provide authenticity and why (not)? And finally, does an authentication scheme provide confidentiality and why (not)?

> Those security notions admit several variants depending of the actual use case they are needed for. As high-level and intuitive definitions, one could say that :
>
> - A scheme E_k using a secret k to transmit a ciphertext $c := E_k(m)$ achieves confidentiality if it is impossible to retrieve the message m from c without knowing the secret k.
> - A scheme S_k using a secret k to transmit a message m achieves authenticity if it is impossible to create a forgery, i.e., a valid pair $(m, s) := S_k(m)$ without knowing the secret k. Authenticity includes integrity in the sense that aftering observing a legitimate valid pair (m, s), it must be impossible to create a different valid pair $(m', s') \neq (m, s)$ without knowing the secret k.
>
> In general, an encryption scheme provides no authenticity. It ensures confidentiality to its users but does not prevent malicious attackers from impersonating them through a *meet-in-the-midle* attack, for instance. As another example, with a stream cipher, an attacker can flip bits of the ciphertext to flip the corresponding bits of the plaintext, even if not knowing their value in the plaintext.
>
> In general, an authentication scheme does not provide confidentiality as it does not transform the message but instead adds a tag (MAC or signature) to it. Usually, the original message is transmitted along with the tag.

■ **Question 2** (12%): 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 $\mathbb{N}$. Let H be a cryptographic hash function. Would the following schemes $E^{(1)}$, $E^{(2)}$ and $E^{(3)}$ be IND-CPA secure?

$$E_k^{(1)}(d, m) = E_k(H(d), m)$$
$$E_k^{(2)}(d, m) = E_k(d, m) \,||\, H(m)$$
$$E_k^{(3)}(d, m) = E_k(H(k||m), m) \,||\, H(k||m)$$

For each scheme, if it is IND-CPA secure, please justify briefly, and specify if there are properties that H needs to satisfy for this. Otherwise, please specify how an adversary can win the IND-CPA game, and how much effort is necessary.

- $E_k^{(1)}$ is still IND-CPA-secure. Indeed, H is assumed to be a cryptographic hash function and thus, it should be collision resistant. This implies that if d is indeed a nonce, $H(d)$ also acts as a nonce and doesn't lead to any vulnerability.
  Note that H must be collision resistant, otherwise the adversary could find $d_1 \neq d_2$ such that $H(d_1) = H(d_2)$, and then win the IND-CPA game by reusing the same diversifier in E_k while respecting the diversifier uniqueness at the level of $E_k^{(1)}$.
- $E_k^{(2)}$ is not IND-CPA-secure. Appending the hash of the message gives us a way to win the IND-CPA game. The strategy is the following :
  1. The adversary randomly chooses two messages m_0 and m_1 and a diversifier d and transmits (m_0, m_1, d) to the challenger.
  2. The challenger responds with $c = E_k^{(2)}(d, m_b) = E_k(d, m_b)||H(M_b)$, with $b \in \{0, 1\}$. Because the specification of E and H are publicly known, the adversary can extract the last bits of c to obtain $H(m_b)$.
  3. The adversary computes $H(m_0)$ and compares it with $H(m_b)$. If there are the same, they outputs $b' = 0$. Otherwise they outputs $b' = 1$.

  Because H is assumed secured, the equality $H(m_b) = H(m_0)$ implies $m_b = m_0$ with overwhelming probably and so we are almost certain to win the game after the first attempt. Note that no query is required, here.
- $E_k^{(3)}$ cannot be IND-CPA-secure because the diversifier has no effect. In order to win the IND-CPA game, we can apply one strategy that works for every scheme without diversification :
  1. The adversary randomly chooses a message m' and a diversifier d' and sends (m', d') as a query. The challenger answers with the ciphertext c'.
  2. The adversary chooses two messages : m_0 equals m' and m_1 is random. They then chooses a diversifier $d \neq d'$ and transmits (m_0, m_1, d) to the challenger.
  3. The challenger responds with a ciphertext $c = E_k(H(k|m_b), m_b) \,|\, H(k|m_b)$.
  4. The adversary checks whether or not the equality $c' = c$ holds. If it does, they output $b' = 0$. Otherwise they output $b = 1$.

  This strategy respects the rules of the game : the adversary never uses the same diversifier twice. Yet, because the encryption only depends on the message and the key (which doesn't change), the equality $E_k^{(3)}(d, m_b) = E_k^{(3)}(d', m')$ implies (with overwhelming probability) $m_b = m'$. And again, we expect to win the game after the first attempt, using a single query.

■ **Question 3** (12%): A hypothetical encryption scheme, called RC404, has been standardized with only the following security claim: *"RC404 resists against key recovery attacks with a security strength of 128 bits in the known plaintext model."*

For each of the following attacks, please state whether it contradicts the claim and why (not).

a. A method that recovers the key with a success probability of one in a billionth ($\approx 2^{-30}$), requires 2^{100} known plaintext-ciphertext pairs and runs in about 2^{100} times the execution time of RC404.

b. A method that recovers with certainty the first byte of the plaintext (although not the next ones) from the ciphertext only, with a running time of a few hours on a modern PC.

c. A method that recovers the key with certainty after seeing the ciphertexts corresponding to the 10! $\approx 2^{22}$ plaintexts composed of the 10 letters "ALGORITHMS" in any order, with a running time of a few hours on a modern PC.

What criticism can you make on RC404 and on its security claim?

Let us start with reading the security claim and identify what is *left unclaimed*:

- other goals than key recovery,
- any attack with higher security strength level than 128 bits, i.e., with running time t, data complexity d and success probability ϵ such that $\log_2(t+d) - \log_2(\epsilon) > 128$, and
- other data models than the known plaintext model.

a. First note that $\log_2(t+d) - \log_2(\epsilon) > 128$ is equivalent to $\frac{t+d}{\epsilon} > 2^{128}$. In this first attack, $t = d = 2^{100}$ and $\epsilon = 2^{-30}$, so $\frac{2^{101}}{2^{-30}} = 2^{131}$, which is greater than 2^{128}. Hence, this attack *does not contradict* the claim as it is above the 128-bit security strength level.

b. The second attack does not recover the key but only some information about the plaintext. While this attack shows that RC404 does not ensure confidentiality properly, it *does not contradict* the claim as only the key recovery goal is claimed.

c. The third attack requires a chosen plaintext data model, as it needs a specific set of plaintext inputs to work. So, it *does not contradict* the claim as only security in the known plaintext model is claimed.

Clearly, this shows that RC404 would be a very bad encryption scheme. First, it does not provide proper confidentiality as the first byte of plaintext can be fairly easily recovered. Second, even the key can be fairly easily recovered, thereby destroying all confidentiality, if the attacker can get access to it for the encryption of a few million specific plaintexts.

The security claim leaves much to be desired as it does not claim security for all goals and all data models. A better claim would be to say that the scheme is indistinguishable from an ideal scheme in the chosen plaintext model (IND-CPA) or in the chosen plaintext and ciphertext model (IND-CCA). The 128-bit security strength level, however, is sufficient for many years to come as it would require from the attacker astronomical resources to go above 128 bits of security.

Secret-key cryptography

Rijndael / AES

■ **Question 4** (6%): What is a block cipher? How do you use it in an encryption or authentication scheme, in general? Even if the best attack against a block cipher is the exhaustive key search, is it enough to guarantee the security of an encryption/authentification scheme that uses this block cipher? Why (not)?

For a block size n and a secret key size m, a block cipher is a mapping $E : \mathbb{Z}_2^n \times \mathbb{Z}_2^m \to \mathbb{Z}_2^n$

- from a secret key $k \in \mathbb{Z}_2^m$ and an input block $x \in \mathbb{Z}_2^n$
- to an output block $y = E_k(x) \in \mathbb{Z}_2^n$.

For each key k, it has an inverse: $x = E_k^{-1}(y)$.

To make an encryption or authentication scheme from a block cipher, it is necessary to specify a *mode of operation* such as CBC, CTR, CBC-MAC, etc.

Even if the best attack against a block cipher is exhaustive key search, an encryption or authentication scheme using it can be broken if the mode of use is bad. For instance, the ECB mode leads to an unsecure encryption scheme despite the quality of the block cipher. One could even imagine pathological modes of operation that would output the secret key as a part of the ciphertext.

■ **Question 5** (10%): Consider two executions of the block cipher AES-128 with the same secret key. The first one has input block X and the second input block X'. If X and X' differ only in *two bytes*, at least how many bytes will differ after two rounds? And at most? Note that the position of the two bytes is not specified, and so you may specify it to be able to reach the minimum or the maximum.

> We must track the bytes that differ between the two executions through the steps of two rounds, namely, SubBytes, ShiftRows, MixColumns, AddRoundKey, SubBytes, ShiftRows, MixColumns, AddRoundKey. First, we remark that SubBytes and AddRoundKey do not modify the location nor the number of bytes that differ, so we can forget them in our reasoning. Second, as ShiftRows only changes the position of the bytes, the initial location of the two differing bytes can be equivalently specified before or after the first ShiftRows. For simplicity, we can specify them after the first ShiftRows. As a result, we can simplify our reasoning and track the differing bytes through the sequence MixColumns, ShiftRows, MixColumns.
>
> After two rounds, the minimum number of differing bytes is 12 and the maximum is 16. Let us see why.
>
> - For the minimum, we consider two cases based on the initial location of the two differing bytes.
>   - If they are in the same column, the MDS property implies that there are at least 3 differing bytes in the column after MixColumns. After ShiftRows, these three bytes are moved to three different columns. Then, MixColumns expands each differing byte in a column to 4 differing bytes, because of the MDS property. As a result, there are 12 differing bytes after two rounds (see case 1 in Figure 1).
>   - If they are in different columns, the MDS property implies that there are 4 differing bytes in these two columns after MixColumns. After ShiftRows, these eight bytes are moved to different columns with two bytes in each. Then, MixColumns expands each two differing bytes in a column to at least 3 differing bytes, because of the MDS property. As a result, there are 12 differing bytes after two rounds (see case 2 in Figure 1).
> - For the maximum, we only need to show one case where 16 bytes can differ. Let us arbitrarily put the two differing bytes in the same column. Then, after the first MixColumns, there can be 4 differing bytes in the same column. These bytes are then moved to different columns after ShiftRows. Finally, MixColumns expands each differing byte in a column to 4 differing bytes, because of the MDS property. As a result, there are 16 differing bytes after two rounds (see case 3 in Figure 1). (Note that we can also reach the maximum of 16 by letting the initial differing bytes be in different columns.)

Hashing

SHA-512/256 is a hash function standardized by the NIST. In a nutshell, it works like SHA-512, except that the output is truncated to 256 bits. In more details, it uses the Merkle-Damgård construction on top of SHA-512's compression function. Hence, it has a chaining value of 512 bits and a message block size of 1024 bits. The output consists of the first 256 bits of the final chaining value. At the time of this writing, there has been no weakness found in its compression function.

■ **Question 6** (10%): In the light of these specifications of SHA-512/256,

a. what is its preimage resistance?

b. what is its second preimage resistance?

c. what is its collision resistance?

d. is producing an internal collision a good strategy to find a collision in this hash function, and why (not)?

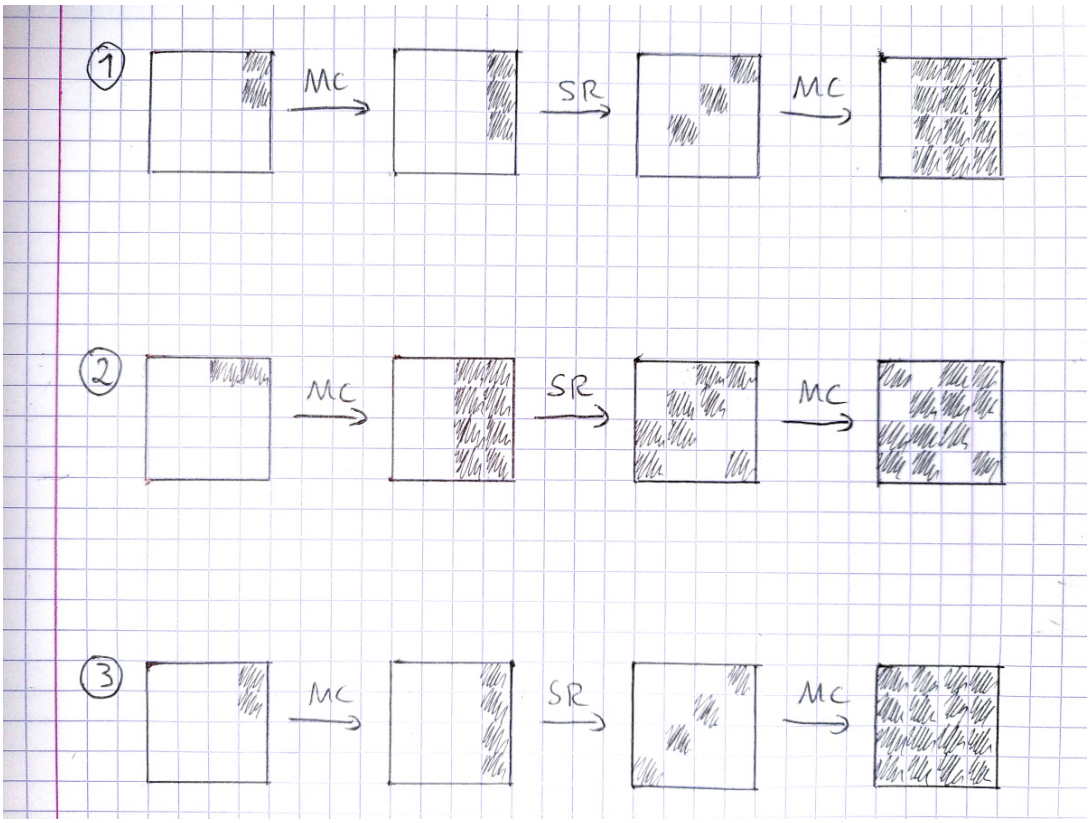


Figure 1

The hash function uses a strong compression function. Also, it has a large enough chaining value, i.e., with 512 bits of chaining value, an internal collision would require 2^{256} attempts. So, in general, it should be suitable for 256-bit security.

a. The preimage resistance is 256 bits as the output size is 256 bits.

b. The second preimage resistance is 256 bits as the output size is 256 bits.

c. Because of the birthday paradox, the collision resistance is 128 bits as the output size is 256 bits.

d. Producing an internal collision is not a good strategy to find a collision in this hash function since the chaining value is longer than the output. So, it would take more effort to search for an internal collision than for an output collision.

■ **Question 7** (10%): For authenticating transactions, we would like to use a message authentication code $\text{MAC}_K(\text{message})$, but we hesitate between different options. Here K is a 128-bit key and the message can be of any size. For each option, please state whether it is secure and briefly justify your answer.

a. $\text{MAC}_K(\text{message}) = \text{SHA-512}(K \| \text{message})$

b. $\text{MAC}_K(\text{message}) = \text{SHA-512/256}(K \| \text{message})$

c. $\text{MAC}_K(\text{message}) = \text{HMAC-SHA-512}_K(\text{message})$

d. $\text{MAC}_K(\text{message}) = \text{HMAC-SHA-512/256}_K(\text{message})$

Which one is the most efficient and secure option in your opinion?

The key point to remember here is length extension attacks (LEAs)!

a. This first option is *not secure* as it is susceptible to LEAs.

b. This second option looks similar to the first one, but it is no longer susceptible to LEAs. SHA-512/256 only outputs the first 256 bits of the chaining value. To apply the length extension attack, the adversary would need to guess the remaining 256 bits, which is more costly than guessing the 128-bit key. So, this option is *secure*.

c. This third option is *secure*, as HMAC is specifically designed to thwart LEAs.

d. The same conclusion goes for this last option, even if the chaining value is shorter (but sufficiently long).

HMAC implies an extra call to the compression function, hence the third and fourth options are slightly less efficient than the second one. So, in our opinion, the second option is the most efficient one while still being sufficiently secure.

Public-key cryptography

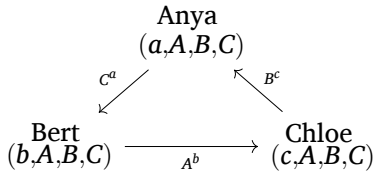
A tripartite key-exchange

Anya, Bert and Chloe want to communicate using AES but to do this, they first need to agree upon a common secret that they could use to derive the AES secret key. They want to use the Diffie-Hellman key exchange but this protocol allows to share a secret between two parties only. The goal of this exercise is to create a variant of Diffie-Hellman in which three people can obtain the same secret.

Here, we fix a prime p and a generator g of the group $\mathbb{Z}_p^*$. Anya's, Bert's and Chloe's private/public key pairs $(\mathsf{SK}, \mathsf{PK})$ are $(a, A = g^a), (b, B = g^b), (c, C = g^c)$ with a, b, c in $[1, \ldots, p-2]$. The reduction modulo p is implicit as we work in $\mathbb{Z}_p^*$.

■ **Question 8** (8%): Propose a protocol based on Diffie-Hellman that would allow all three people to share the common secret g^{abc}. *Hint:* The parties can first work in pairs. What is the cost of your protocol compared to the original Diffie-Hellman protocol ?

One simple solution is the key exchange described by the following diagram. The variables in parenthesis are the data available by each participant.



The step-by-step protocol is as follow :

1.
   - Anya computes C^a and sends it to Bert.
   - Bert computes A^b and sends it to Chloe.
   - Chloe computes B^c and sends it to Anya.

2.
   - Anya computes $(B^c)^a = g^{abc}$.
   - Bert computes $(C^a)^b = g^{abc}$.
   - Chloe computes $(A^b)^c = g^{abc}$.

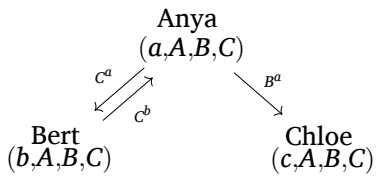
Note that everyone's computations involve only data available to them. In this scheme, 3 modular exponentiations are computed in the first round and 3 others in the second round, for a total of 6 exponentiations. Considering a usual Diffie-Hellman protocol requires a total of 2 exponentiations [1] , the computational cost of our proposed protocol is threefold.

Variants of the proposed scheme are also possible. For instance, we could do the following :

1.
   - Anya computes C^a and sent it to Bert.
   - Anya computes B^a and sent it to Chloe.
   - Bert computes C^b and sent it to Anya (or Chloe computes B^c and send it to Anya)

2.
   - Anya computes $(C^b)^a = g^{abc}$ (or $(B^c)^a$ if Chloe sent her B^c)
   - Bert computes $(C^a)^b = g^{abc}$.
   - Chloe computes $(B^a)^c = g^{abc}$.

We can represent this by the following diagram



This scheme allows everyone to obtain the shared secret with a total of 6 exponiations just like the previous solution but it is not as elegant as the first one as it introduces an asymmetry [2] between the three parties.

Generic group notation

Let G be a group of order $|G| = q$ and g is a generator of this group.

■ **Question 9** (6%): Rewrite the two-party Diffie-Hellman algorithm in the generic group notation so that it can be applied, e.g., to elliptic curves. What is the expression of the common secret with the generic group notation?

> With the generic group notations, the respective key pairs of Anya and Bert are $(a, A = [a]g)$ and $(b, B = [b]g)$ with a generator g of G and $a, b \in [1, q-1]$.
> The usual Diffie-Hellman key exchange then consists of the following steps :
>
> 1. • Anya sends A to Bert
>   • Bert sends B to Anya
>
> 2. • Anya computes the shared secret $[a]B = [a][b]g = [ab]g$
>   • Bert computes the shared secret $[b]A = [b][a]g = [ab]g$

A one-round tripartite Diffie-Hellman using a pairing

We now want to improve our protocol and improve its efficiency. In order to do this, we will use a special kind of function (often used in elliptic curve cryptography) called a *pairing*.

Let E be an elliptic curve, G the group of points on this curve and P a point that is a generator of the group G. Let $q = |G|$ be the order of G, i.e., the number of elements in G. Anya's, Bert's and Chloe's private/public key pairs $(\mathsf{SK}, \mathsf{PK})$ are now $(a, A = [a]P), (b, B = [b]P), (c, C = [c]P)$ where a, b, c are integers in $\mathbb{Z}_q$.

Finally, we define a map $F : \mathbb{Z}_q \times G \times G \to \mathbb{Z}_q^*$ with the following properties. If P is a generator of the group G, then for any points R and Q and any integer x in $\mathbb{Z}_q$, we have

- $F(1, P, P) = u$, where u is some fixed and known element in $\mathbb{Z}_q^*$,
- $F(x, R, Q) = F(1, R, Q)^x$,
- $F(1, [y]R, Q) = F(1, R, Q)^y$,
- $F(1, R, [z]Q) = F(1, R, Q)^z$.

Such a map F is what we call a *pairing*. (*Note:* you can safely assume the pairing F to be already defined and use it abstractly.)

In essence, these properties allow us to compute $F(x, Y, Z)$ for any point $Y = [y]P$ and $Z = [z]P$, and the result can be expressed as a power of the known integer u. Note that every point can be expressed as a multiple of P since we assume P is a generator.

■ **Question 10** (6%): Compute the value of $F(1, A, B)$ and $F(a, P, P)$.

> We simply apply the formulas :

1.

$$\begin{aligned} F(1,A,B) &= F(1,[a]P,[b]P) && \text{(by definition of } A \text{ and } B\text{)} \\ &= F(1,P,[b]P)^a && \text{(using the second property of } F \\ &= F(1,P,P)^{ab} && \text{(using the third property of } F\text{)} \\ &= u^{ab} && \text{(by definition of } u\text{)} \end{aligned}$$

2.

$$\begin{aligned} F(a,P,P) &= F(1,P,P)^a && \text{(using the first property of } F\text{)} \\ &= u^a && \text{(by definition of } u\text{)} \end{aligned}$$

■ **Question 11** (10%): Propose a protocol in which each party obtain a common secret by applying the pairing F only once. What is the expression of this common secret?

If Anya, Bert and Chloe have respective key pairs $(a, A = [a]P)$, $(b, B = [b]P)$ and $(c, C = [c]P)$, the tripartite key exchange simply consist of one step :

- Anya computes $F(a,B,C) = F(a,P,P)^{bc} = F(1,P,P)^{abc} = u^{abc}$
- Bert computes $F(b,A,C) = F(b,P,P)^{ac} = F(1,P,P)^{abc} = u^{abc}$
- Chloe computes $F(c,A,B) = F(c,P,P)^{ab} = F(1,P,P)^{abc} = u^{abc}$

Then can therefore use u^{abc} as the shared secret. This solution has several advantages compared to the one proposed for question 8 : it only requires one round of computation instead of two and more importantly, it is *non-interactive*. As long as everyone's public key is made public (which is only done once), any group of three people can compute such a shared secret without having to send and receive intermediate messages from the others.
This idea was presented in 2000 by Antoine Joux, in the paper *A One Round Protocol for Tripartite Diffie–Hellman*.

Portfolio

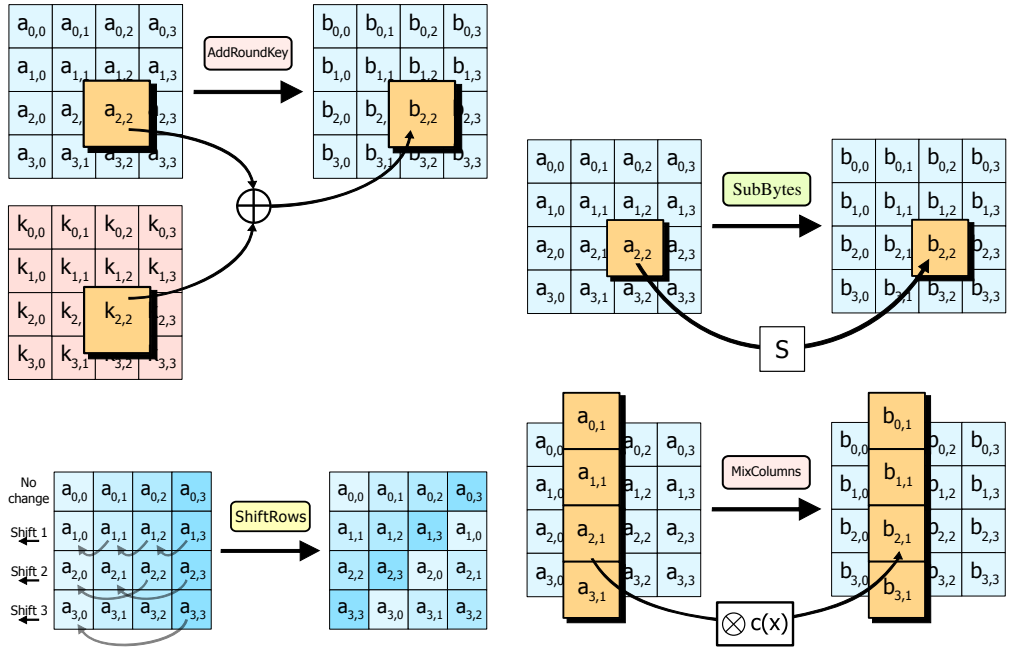
IND-CPA (chosen plaintext, chosen diversifier)

A scheme $\mathcal{E} = (\text{Gen}, \text{Enc}, \text{Dec})$ is **IND-CPA-secure** if no adversary can win the following game for more than a negligible advantage.

1. Challenger generates a key (pair) $k \leftarrow \text{Gen}()$
2. Adversary queries Enc_k with (d, m) of his choice
3. Adversary chooses d and two plaintexts $m_0, m_1 \in M$ with $|m_0| = |m_1|$
4. Challenger randomly chooses $b \leftarrow_R \{0,1\}$, encrypts m_b and sends $c = \text{Enc}_k(d, m_b)$ to the adversary
5. Adversary queries Enc_k with (d, m) of his choice
6. Adversary guesses b' which plaintext was encrypted
7. Adversary wins if $b' = b$ (Advantage: $\epsilon = \left|\Pr[\text{win}] - \frac{1}{2}\right|$.)

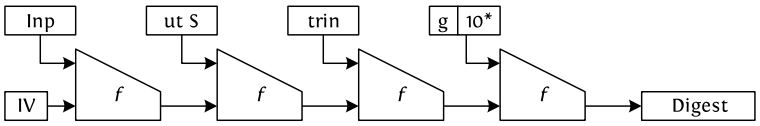
The adversary **must** respect that d **is a nonce**! The values of d used in steps 2, 3 and 5 must all be different.

The step mappings inside Rijndael/AES



MDS property: If N bytes change at the input of MixColumns and as a result M bytes change at its output, then either $N = M = 0$ or $N + M \geq 5$.

The Merkle-Damgård construction



HMAC (simplified)

$$\text{HMAC-}H_K(\text{message}) = H(K \| H(K \| \text{message}))$$

The Diffie-Hellman protocol

In the multiplicative group setting, the public parameters are the group $G = \mathbb{Z}_p^*$ (with p prime) and a generator g.
The keypairs $(\mathsf{SK}, \mathsf{PK})$ of Alice and Bob are respectively $(a, A = g^a)$ and $(b, B = g^b)$, for a, b in $\mathbb{Z}_p^*$.
The key exchange is then described by the following algorithm :

1. Alice sends A to Bob ; Bob sends B to Alice
2. Alice computes B^a ; Bob computes A^b
3. Alice outputs B^a as the shared secret ; Bob outputs A^b as the shared secret