Université Libre de Bruxelles
Faculté des Sciences

January 2024

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 4 questions and includes a portfolio at the end. Please ensure that we can split your answers to questions 1-2 from those of question 3 from those of question 4 by using separate sheets of paper.

■ **Question 1** (20%): Please complete the following sentences on the left with one of the possible right hand sides. Each right hand side may only be used at most once. If two (or more) right hand sides are correct, you may choose one arbitrarily. Please write your answer as digit-letter pairs, e.g., 1*x* 2*y* 3*z* …

1. 128 bits
2. 256 bits
3. 3072 bits
4. A generic attack
5. AES256-CTR
6. An attack in the known plaintext model
7. Diffie-Hellman
8. HMAC-SHA-512
9. MixColumns
10. ShiftRows
11. SubBytes
12. The ElGamal signature scheme
13. The one-time-pad

a. becomes insecure if the diversifier value is reused
b. becomes insecure if the ephemeral key is reused
c. becomes insecure in the known plaintext model
d. is a bit transposition
e. is a block cipher
f. is a key agreement scheme
g. is a linear function that provides diffusion
h. is a non-linear function
i. is a sufficiently large size for a secure RSA modulus
j. is a sufficiently large size for the chaining value of a secure hash function
k. is a typical limit for the plaintext size
l. is a typical security strength level nowadays
m. is an authentication scheme
n. is an elliptic curve
o. is an unconditionally secure encryption scheme
p. is applicable to a mode irrespective of the primitive
q. is applicable to a primitive irrespective of the mode
r. is susceptible to length extension attacks
s. works also in the ciphertext-only model
t. works also in the chosen-plaintext model

The expected sentences were:

1. 128 bits **is a typical security strength level nowadays**
2. 256 bits **is a sufficiently large size for the chaining value of a secure hash function**
3. 3072 bits **is a sufficiently large size for a secure RSA modulus**
4. A generic attack **is applicable to a mode irrespective of the primitive**
5. AES256-CTR **becomes insecure if the diversifier value is reused**
6. An attack in the known plaintext model **works also in the chosen-plaintext model**
7. Diffie-Hellman **is a key agreement scheme**
8. HMAC-SHA-512 **is an authentication scheme**
9. MixColumns **is a linear function that provides diffusion**
10. ShiftRows **is a bit transposition**
11. SubBytes **is a non-linear function**
12. The ElGamal signature scheme **becomes insecure if the ephemeral key is reused**
13. The one-time-pad **is an unconditionally secure encryption scheme**

Note that this is the only combination where all 13 answers are correct. It was possible to have other locally correct answers, but then at the cost of "stealing" a right hand side that would otherwise belong elsewhere. Specifically:

- 256 bits are also a typical security strength level nowadays, but 128 bits are not sufficient for a chaining value nor for a RSA modulus.
- 3072 bits are sufficient (and overkill) for a chaining value, but 256 bits are not sufficient for a RSA modulus.
- The ElGamal signature scheme is an authentication scheme, but then there was no possible right hand side for HMAC-SHA-512.

As a common mistake, note that AES256-CTR is not a block cipher. It is the combination of a block cipher and a mode, and in this case it is an encryption scheme.

■ **Question 2** (20%): 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 $\{0,1\}^{128}$, the plaintext space to be arbitrary and the diversifier space be the set of n-bit strings $\{0,1\}^n$ for some fixed public parameter value n. Let H be a cryptographic hash function that outputs n bits.

a. As (Gen, E, D) is an "IND-CPA secure symmetric-key encryption scheme with diversification", what condition on the diversifier d the user must ensure for the scheme to be secure?

> The condition is that the value of d must be unique upon each encryption for a given key.
>
> Note that, when the same value of d is used to encrypt two or more plaintexts:
>
> - In general, identical plaintexts can be recognized from identical ciphertexts, thereby breaking IND-CPA.
> - For stream chiphers specifically, the bitwise difference between two plaintexts is revealed as it is equal to the bitwise difference between the two ciphertexts. This gives more information about the plaintext than just whether two are equal or not.

The user has access to a source of perfectly random bits and would like to use these to convert the scheme into a randomized one. More precisely, s.he has access to the function `random()` that returns r bits of randomness, for some fixed public parameter value r. So, s.he defines a new encryption scheme (Gen', E', D') that works as follows:

- Gen' calls Gen unmodified;
- $E'_k(m)$ calls $R \leftarrow$ `random()` and returns $E_k(H(R), m) \| R$, i.e., the diversifier is replaced with the hash of a random value and the random value is attached to the ciphertext.

Then, we ask the following questions.

b. How does the decryption D' work?

> Upon receiving a ciphertext c, the decryption algorithm D' can proceed as follows:
>
> - It first removes R from the ciphertext c by taking the last r bits. Let c' be the ciphertext after removing R. (Since r is a fixed public parameter, there is no ambiguity in how to split c into $c' \| R$.)
> - It then computes $H(R)$.
> - Finally, it calls the original decryption algorithm D with the truncated ciphertext c' and with $H(R)$ as diversifier, $D_k(H(R), c')$.

c. What are the conditions on the parameter values n and r for the randomized scheme to be IND-CPA secure at a strength level of about 128 bits? Why?

> We must have $n \geq 256$ and $r \geq 256$.
>
> As noted in the first subquestion, the diversifier must be unique for each encryption. In this randomized algorithm, the diversifier is randomly drawn and computed as $H(R)$. There are two cases where $H(R)$ would repeat:
>
> - Two values of the random R are equal. Since a hash function like H is deterministic, in this case $H(R)$ would be equal too.
> - Two different values of the random $R_1 \neq R_2$ evaluate to identical outputs $H(R_1) = H(R_2)$. This would exhibit a collision in H.

The required security strength level of 128 bits implies that nothing bad should happen with significant probabilty with less than 2^{128} encryptions. (Note: to see this, say that the data complexity is $d < 2^{128}$ and that the probability is almost one $\epsilon \approx 1$. Then, $\log_2 d - \log_2 \epsilon \approx \log_2 d < 128$, contradicting the requirement that this should be ≥ 128.)

Because of the birthday paradox, the first event can occur with significant probabilty after $2^{r/2}$ draws of R, so we need $2^{r/2} \geq 2^{128}$, which in turn means that $r \geq 256$ bits.

Similarly, the second event can occur with significant probability after $2^{n/2}$ evaluations of $H(R)$ with different R's. Hence, we must have $n \geq 256$ bits too.

d. Is the use of the hash function H really necessary? Why (not)?

No, the hash function is not really necessary.

The only required property that we want for the diversifier is that it is unique at each encryption. The diversifier does not have to be secret—it can be public, and it is so here (part of the ciphertext). If we use random numbers of sufficient size to avoid repetitions, hashing them brings no added value. We could just have defined $E'_k(m)$ as $E_k(R, m)\|R$, and it would have been at least as secure.

■ **Question 3** (25%): NIST is currently in the process of standardizing Ascon. When designed, Ascon was only an authenticated encryption scheme, but NIST plans to also include Ascon-Hash, an extendable output function (XOF) based on it. Specifically, Ascon-Hash is a sponge function that uses the same 320-bit permutation f_{Ascon} as Ascon itself.

a. Please explain what we mean with "320-bit permutation f_{Ascon}".

   This means that f_{Ascon} is a bijective mapping $\{0,1\}^{320} \to \{0,1\}^{320}$.

b. NIST aims for 128-bit security on all attacks. What is the minimum value of the capacity c and why?

   A sponge function with a capacity c resists against generic attacks up to a complexity of $2^{c/2}$. So, to achieve 128-bit security, one needs $c \geq 2 \times 128 = 256$ bits.

c. Let us now assume that c is the minimum value found in the previous point. What is the rate r of the Ascon-Hash sponge function?

   In the sponge construction, the size of the state b equals the number of input/output bits of the underlying permutation, so $b = 320$. The state is composed of the outer part containing r bits and the inner part containing c bits, hence $b = r + c$. Since $c = 256$ from the subquestion above, we have $r = b - c = 320 - 256 = 64$ bits.

d. Consider a hypothetical competing sponge function XYZ-Hash that uses a permutation f_{XYZ} of 384 bits instead, but otherwise uses the same capacity c as above. Assume that the evaluation of a permutation takes a time that is proportional to its size, i.e., say that the evaluation of f_{Ascon} takes 320 and that of f_{XYZ} takes 384 in some arbitrary time unit. Furthermore, assume that this time dominates over the other operations in the sponge construction. Which of Ascon-Hash or XYZ-Hash is going to be faster to evaluate in general? What is the throughput ratio between Ascon-Hash and XYZ-Hash? (Here, no need to compute the exact numerical value, writing the expression is enough.)

   If we keep $c = 256$ for both functions, XYZ-Hash's rate is $r_{\text{XYZ}} = 384 - 256 = 128$ bits.

   Looking at the definition of the sponge construction, the rate tells us how many bits are input (absorbed) or output (squeezed) per evaluation of the permutation, which is the dominating cost. Hence, since $r_{\text{XYZ}} = 128$ and $r_{\text{Ascon}} = 64$, XYZ-Hash processes twice more bits per evaluation of their respective permutation than Ascon.

   However, this does not tell us everything, because f_{XYZ} takes more time to evaluate than f_{Ascon}. We can already see that f_{XYZ} takes less than twice the amount of time than f_{Ascon}, so the net effect is that XYZ-Hash will still process more bits per time unit than Ascon.

   To evaluate this more precisely, we note that Ascon processes $r_{\text{Ascon}} = 64$ bits per 320 time units, and XYZ-Hash $r_{\text{XYZ}} = 128$ bits per 384 time units. The throughput ratio is therefore:

   $$\frac{\frac{64}{320}}{\frac{128}{384}} = \frac{3}{5}\,.$$

   Overall, XYZ-Hash is therefore 5/3 times faster than Ascon.

e. Consider an adversary who would like to find a string x such that Ascon-Hash(x)'s first 80 output bits are all 1. How can s.he proceed? How many evaluations of f_{Ascon} do you expect it will take before succeeding?

   This is a preimage attack on a hash function with 80 bits of output. The attacker can proceed by trying arbitrary values x, evaluating Ascon-Hash(x) and checking if the first 80 output bits are all 1 as required. This is going to take about 2^{80} attempts, as each attempt succeeds with a probability of $1/2^{80}$.

   This is not all, because this tells us how many evaluations of Ascon-Hash(x) we need to make, but not how many evaluations of f_{Ascon} this implies. This leads to two questions:

- How many calls to f_{Ascon} per evaluation of Ascon-Hash do we need to absorb a candidate x?
- How many calls to f_{Ascon} per evaluation of Ascon-Hash do we need to squeeze 80 output bits?

To address the first question, we must know the size of x. In this attack, we are free to take candidates x of any size. However, if we restrict ourselves to too short strings, we will soon run out of possibilities before we can complete our expected 2^{80} attempts. E.g., if we restrict ourselves to strings of at most, say, 30 bits, we can only make $2^{31} - 1$ different candidates x. In other words, no matter how we choose our candidates x, the vast majority of them will have a length of about 80 bits or more. So, for simplicity, let us assume that we try all possible strings x in $\mathbb{Z}_2^{80}$.

How much does it cost to absorb a 80-bit string x? Since Ascon-Hash's rate is $r = 64$ bits, we need to absorb x as two blocks, hence it takes 2 calls to f_{Ascon}.

Regarding the second question, we need to squeeze 80 bits of output. Just after absorbing x, we have $r = 64$ bits available immediately. To get the remaining 16 bits, we need to call f_{Ascon} one more time.

Hence, to absorb a 80-bit x and squeeze 80 bits of output with Ascon-Hash, we need to call f_{Ascon} 3 times. So, we expect that this preimage attack will take about 3×2^{80} evaluations of f_{Ascon} before it succeeds.

(If you reached this point correctly, you received the maximum grade for this subquestion.)

(Bonus points if your procedure is optimized.)

How can we optimize the attack procedure to reduce the cost below 3×2^{80} evaluations of f_{Ascon}?

The first optimization [some students had it] addresses the squeezing phase. Just after absorbing x, we have $r = 64$ bits available immediately. If these 64 bits do not have the desired all-1 value, there is no use in calling f_{Ascon} to produce 16 more. Only if these 64 bits are all-1, then we check further, and this happens with a probability of 2^{-64}. So, with this idea, we now have a cost of about $(2 + 2^{-64}) \times 2^{80}$ evaluations of f_{Ascon}.

The second optimization addresses the absorbing phase. For simplicity of the description, let us ignore padding and assume that all the candidates x have 128 bits so that they span exactly 2 blocks of $r = 64$ bits. We split x into two blocks of 64 bits, $x = x_1 \| x_2$. For all the strings x that share the same first block x_1, the state of the sponge function after absorbing x_1 is the same: $s_1 = f_{\text{Ascon}}(x_1 \| 0^c)$. Our strategy now consists in evaluating many candidates x that share the same first half, computing s_1 only once, and continuing with absorbing the 2^{64} possible values of x_2. We change x_1 (and so s_1) only when the 2^{64} possible second blocks have been tested. This reduces the cost of the attack down to about $(1 + 2^{-64} + 2^{-64}) \times 2^{80} \approx 2^{80}$ evaluations of f_{Ascon}.

To sum up, the optimized attack works as follows:

```
• For each $x_1 \in \mathbb{Z}_2^{64}$:
    – $s_1 \leftarrow f(x_1 \| 0^c)$ [absorb $x_1$]
    – For each $x_2 \in \mathbb{Z}_2^{64}$:
        ⋆ $s_2 \leftarrow f(s_1 \oplus (x_2 \| 0^c))$ [absorb $x_2$]
        ⋆ $z \leftarrow$ the first 64 bits of $s_2$ [squeeze 64 bits]
        ⋆ If $z = 1^{64}$:
            · $s_3 \leftarrow f(s_2)$
            · $z \leftarrow$ the first 16 bits of $s_3$ [squeeze 16 more bits]
            · If $z = 1^{16}$: output $x_1 \| x_2$ as solution, and stop. Success!
```

The line "If $z = 1^{64}$" will be executed 2^{80} times to test 2^{80} candidates. The number of times f_{Ascon} is evaluated can be deduced from there:

- $s_2 \leftarrow f(\dots)$ will be computed 2^{80} times, once per test;
- $s_3 \leftarrow f(\dots)$ will be computed 2^{16} times, only when $z = 1^{64}$;
- $s_1 \leftarrow f(\dots)$ will be computed 2^{16} times, once per 2^{64} inner iterations.

■ **Question 4** (35%): Alice and Bob would like to communicate with each other confidentially. In the following, we describe a protocol based on RSA that Alice and Bob plan to use to establish a secret key that they will subsequently use for symmetric encryption. However, there are mistakes (possibly omissions) in this protocol. Some of them are functional, that is, they prevent the protocol from working correctly, while others introduce security flaws.

Public key setup at Alice's side

1. Alice generates two random prime numbers p_A and q_A of 1536 bits each.
2. She computes:
   - $n_A = p_A q_A$
   - $e_A = 4$
   - $d_A = e_A^{-1} \bmod (p_A - 1)(q_A - 1)$
3. She sends (n_A, e_A) to Bob.

Public key setup at Bob's side

4. Bob generates two random prime numbers p_B and q_B of 1536 bits each.
5. He computes:
   - $n_B = p_B q_B$
   - $e_B = 3$
   - $d_B = e_B^{-1} \bmod (p_B - 1)(q_B - 1)$
6. He sends (n_B, e_B) to Alice.

Key establishment

Let H be a secure hash function that outputs 256 bits.

7. Alice chooses a random string m of 512 bits.
8. Alice computes the 256 bits of output of $H(m)$ and interprets them as the integer k with $0 \leq k \leq 2^{256} - 1$.
9. Alice encrypts k as $c = k^{e_B} \bmod n_B$.
10. Alice signs m as $s = m^{d_A} \bmod n_A$.
11. Alice sends (c, s) to Bob.
12. Bob decrypts the secret key that Alice chose: $k = c^{d_B} \bmod n_B$.
13. Bob verifies the signature by computing $m' = s^{e_A} \bmod n_A$ and then ensuring that $k = H(m')$; otherwise he aborts.
14. Both parties use k as their shared secret key.

The protocol is summarized in Figure 1, where the elements listed below each party's name correspond to the data available to them.

We now ask the following questions :

Figure 1

Alice	Eve	Bob
$(n_{\mathrm{A}}, e_{\mathrm{A}}, d_{\mathrm{A}}), (n_{\mathrm{B}}, e_{\mathrm{B}})$	(?)	$(n_{\mathrm{B}}, e_{\mathrm{B}}, d_{\mathrm{B}}), (n_{\mathrm{A}}, e_{\mathrm{A}})$
Randomly chooses m Computes $k = H(m)$		
Computes $c = k^{e_{\mathrm{B}}} \bmod n_{\mathrm{B}}$ Computes $s = m^{d_{\mathrm{A}}} \bmod n_{\mathrm{A}}$	$\xrightarrow{(c,s)}$	
		Computes $k = c^{d_{\mathrm{B}}} \bmod n_{\mathrm{B}}$ Computes $m' = s^{e_{\mathrm{A}}} \bmod n_{\mathrm{A}}$
		Computes $H(m')$ and checks it is equals to k; if not, aborts
(Uses k as a secret key for symmetric encryption)		*(Uses k as a secret key for symmetric encryption)*

a. On which mathematical problem does the security of RSA rely?

RSA relies on the difficulty of the factorization problem, i.e., the problem of factoring a large number into its prime factors.

Note: If Eve, an eavesdropper, can factor, say, n_{A} into $n_{\mathrm{A}} = p_{\mathrm{A}} \times q_{\mathrm{A}}$, she can compute the private exponent d_{A} just like Alice did. In the other direction, if there exists an efficient algorithm to recover d_{A} from $(n_{\mathrm{A}}, e_{\mathrm{A}})$, this can be used to factor n_{A}. Therefore, any advance in factorization is an advance in recovering a private RSA key, and vice-versa. (However, this is only about recovering the private key. More generally decrypting an RSA ciphertext without knowing the private key does not necessarily imply factoring n_{A} as far as we currently know.)

b. In Alice's and Bob's key setup, we (correctly) compute the private exponent d as a multiplicative inverse modulo $(p-1)(q-1)$. Why isn't it computed as a multiplicative inverse modulo $n = pq$ more simply instead?

First, recall that $(p-1)(q-1)$ equals $\varphi(n)$. If we computed d as $e^{-1} \bmod n$ instead of $e^{-1} \bmod \varphi(n)$, the scheme would simply not be correct : the decryption algorithm would not retrieve the correct message m.

What we want from the private and public exponents d and e is that they define operations that are inverse of each other, that is, if $c = m^e \bmod n$, we can recover $m = c^d \bmod n$. In other words, we want that $(x^e)^d \equiv (x^d)^e \equiv x^{ed} \equiv x \pmod{n}$ for any x.

Euler's theorem tell us that when working modulo n, all exponents can be reduced modulo $\varphi(n)$. In our case this implies :

$$m^{ed} \equiv m^{ed \bmod \varphi(n)} \pmod{n}.$$

And because we picked d such that $ed \equiv 1 \bmod \varphi(n)$, the decryption algorithm computes

$$m^{ed} \equiv m^{ed \bmod \varphi(n)} \equiv m^1 \equiv m \pmod{n}$$

and we retrieve the correct message. This would not work if we reduced the exponent ed modulo n instead.

c. What are the general techniques to prevent or detect that a malicious adversary sends its public key to Alice pretending it's Bob's or vice-versa? In particular, we saw two techniques; please name them and briefly explain the common idea they use.

The two techniques we saw are called public-key infrastructure (PKI) and web of trust. In both cases, one binds a public key and its owner's identity by signing both in the same message.

d. Assume that Alice has Bob's genuine public key and vice-versa, and that their respective private keys remain private. Can someone pretend to be Alice and interact with Bob to establish a shared secret key with him? Why (not)?

No. For Bob not to abort, we need to have $H(m') = k$, and since $k = H(m)$ this requires that $m = m'$ (unless H is not collision resistant, but we assumed that H is secure). As $m' = s^{e_A} \bmod n_A$, only the owner of d_A, Alice, could have computed s.

It is also impossible to mount a man-in-the-middle attack because we assumed that Alice already possesses Bob's genuine key (and vice-versa).

e. Alice's key setup is functionally flawed and fails every time. What property is not satisfied in Alice's key setup? Please propose a correction. (Hint: p and q are odd numbers.)

In Alice's setup, we need to compute the multiplicative inverse of $e_A = 4$ modulo $(p_A-1)(q_A-1)$. However, a multiplicative inverse exists only if the value to invert and the modulus are relatively prime (i.e., they share no common factor). But both e_A and $(p_A - 1)(q_A - 1)$ are even, so $d_A = e_A^{-1} \bmod (p_A - 1)(q_A - 1)$ cannot be computed. In other words, there exists no exponent d_A that can invert an exponentiation with the even exponent e_A.

Alice should use an odd public exponent (and also check that e_A has no other common factor with $(p_A - 1)(q_A - 1)$, see below). In practice, common choices for the exponent are $e_A = 3$, $e_A = 2^4 + 1 = 17$ and $e_A = 2^{16} + 1 = 65537$.

f. Bob's key setup is functionally flawed too, but fails only from time to time. Please explain why. What is, approximately, the probability of failure?

The computation $d_B = e_B^{-1} \bmod (p_B - 1)(q_B - 1)$ can be carried out only if $(p_B - 1)(q_B - 1)$ has no common factor with $e_B = 3$, that is, if neither $(p_B - 1)$ nor $(q_B - 1)$ is a multiple of 3.
A quick computation gives :

$$\begin{aligned}
\Pr(e_B = 3 \text{ is coprime with } (p_B - 1)(q_B - 1)) &= \Pr(3 \text{ is coprime with } p_B - 1)\Pr(3 \text{ is coprime with } q_B - 1) \\
&= (1 - \Pr(p_B - 1 \text{ is a multiple of } 3)\ (1 - \Pr(q_B - 1 \text{ is a multiple of } 3) \\
&= (1 - \tfrac{2}{3})(1 - \tfrac{2}{3}) \\
&= \tfrac{4}{9}
\end{aligned}$$

The probability that a valid secret key d_B exists is therefore $4/9$. In other words, the probability of failure is $5/9$.

Note: in practice, one usually first chooses e_B and then looks for secret primes p_B and q_B that do not share a common factor with e_B.

g. List all the data elements known to an eavesdropper, Eve, either because they are public information or because she can intercept them.

Eve has access to all public keys n_A, e_A, n_B, e_B like anyone. We also assume she can see the values of c and s as they are being sent from Alice to Bob via a public channel.

h. The protocol has two security flaws. The first one is that Eve can easily recover k from c without knowing Bob's secret key. Please explain why and propose a correction.

The protocol is not secure against small-exponent attacks.

Specifically, since $k < 2^{256}$ and $e_B = 3$, we have that k^3 (as a plain integer, without modulo) is smaller than $2^{256\times 3} = 2^{768}$. On the other hand, n_B is the product of two 1536-bit primes, so it is of the order of 2^{3072}. Hence, $k^3 < n_B$ and reducing k^3 modulo n_B has no effect, i.e., $c = k^3 \bmod n_B = k^3$. To recover k, it suffices to compute the cubic root of c.

There are many possible ways to correct this.

- Of course, the safest and most straightforward way to fix this is to use a standard approach like RSA-OAEP to encrypt k.
- For a more "local" fix, one could choose a larger e_B, for instance, randomly in the range $1 < e_B < \varphi(n_B)$ so that k^{e_B} becomes much larger than n_B and the reduction modulo n_B does something non-trivial.
- Alternatively, keeping $e_B = 3$, one could increase the value that is raised to the power e_B, for instance, by computing c as $c = (k + r \times 2^{256})^{e_B} \bmod n_B$ with r a random value in the range $0 < r < n_B/2^{256}$, i.e., such that $k + r \times 2^{256}$ contains k in the least significant bits and garbage random bits in the most significant bits (those can be ignored upon decryption).

i. The second security flaw is again that Eve can easily recover k, but from s this time. Please explain why and propose a possible correction.

This is a rather silly mistake.

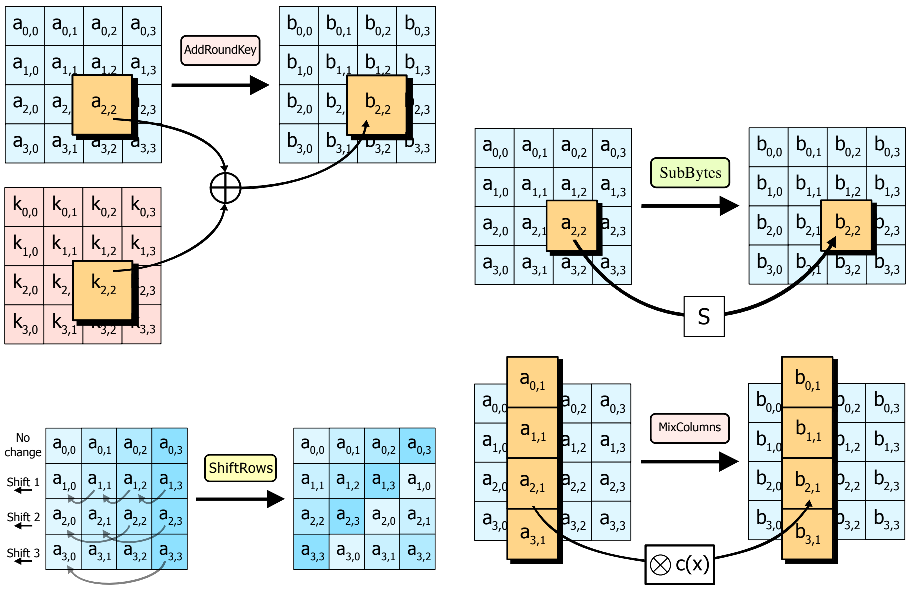
Eve sees s and knows Alice's public key. She can therefore compute $m' = s^{e_A} \bmod n_A$ just like Bob. Assuming that the protocol was run honestly and without other interferences, $m = m'$ and Eve can compute $k = H(m')$.

It is important to keep in mind that a signature, in general, does not ensure any form of confidentially and does not try to hide what is signed in any way. So, signing m is clearly not a good idea, as it contains enough information to compute k. To ensure that Bob is indeed receiving genuine information from Alice, she could sign the ciphertext c instead of m. This way, there is no risk that the signature reveals more than the ciphertext (which Eve already knows anyway).

Of course, the safest and most straightforward way to fix this is to use a standard approach like RSA-PSS to sign c. Alternatively, the simplest "local" fix would be to compute s as $s = H(c)^{d_A} \bmod n_A$, with H preferrably a XOF to use full-domain hashing.

Portfolio

The step mappings inside Rijndael/AES



IND-CPA (chosen plaintext, chosen diversifier)

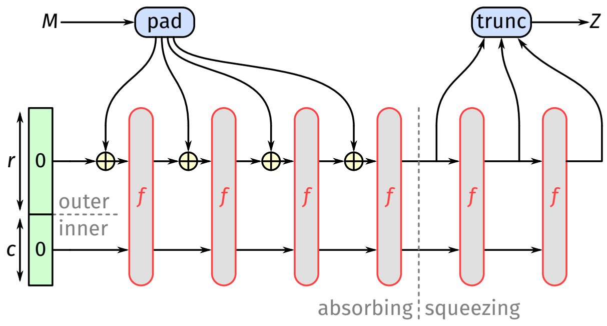
A scheme $\mathcal{E} = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ is **IND-CPA-secure** if no adversary can win the following game for more than a negligible advantage.

1. Challenger generates a key (pair) $k \leftarrow \mathrm{Gen}()$
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 $c = \mathrm{Enc}_k(d, m_b)$ 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: $\epsilon = \left|\Pr[\mathrm{win}] - \frac{1}{2}\right|$.)

Sponge function

The sponge construction is a mode on top of a permutation. It calls a b-bit permutation f, with $b = r + c$, i.e., r bits of *rate* and c bits of *capacity*. A sponge function is a concrete instance with a given f, r, c and implements a mapping from $M \in \{0,1\}^*$ to $\{0,1\}^\infty$ (truncated at an arbitrary length). The differentiating advantage of a random sponge from a random oracle is at most $t^2/2^{c+1}$, with t the time complexity in number of calls to f.

- $s \leftarrow 0^b$
- $M||10^*1$ is cut into r-bit blocks
- For each M_i do
  - $s \leftarrow f(s \oplus (M_i||0^c))$
- As long as output is needed do
  - Output the first r bits of s
  - $s \leftarrow f(s)$


M
pad
trunc
Z
r
c
0
0
outer
inner
f
f
f
f
f
f
absorbing
squeezing