Cryptography and Security — Midterm Exam

Solution

Serge Vaudenay

15.10.2025

- duration: 1h45
- **no documents** allowed, except one 2-sided sheet of handwritten notes
- a pocket calculator is allowed
- communication **devices are not allowed**
- the exam invigilators will **not answer** any technical question during the exam
- readability and style of writing will be part of the grade
- answers should **not be written with a pencil**

The exam grade follows a linear scale in which each question has the same weight.

1 Nonce-Based Multi-Use Perfect Secrecy

We define a nonce-based encryption scheme as follows: given a plaintext X, a key K, and an additional input called *nonce* N, the ciphertext Y is obtained by $Y = \mathsf{Enc}(K, N, X)$. Decryption takes input K, N, Y and returns $X = \mathsf{Dec}(K, N, Y)$. It is forbidden to use the same nonce N more than once to encrypt. Furthermore, we do not assume that the nonce is private (hence, the adversary has access to Y and N). The probability distributions of K, N and X are supposed to be independent. We assume that X, Y, and N belong to the same domain $\mathbf{F}$, which is a finite field. We assume that the distribution of K is determined by the encryption system. We say that the encryption system is *1-time secure* if for any $x, x', y, n \in \mathbf{F}$, we have

$$\Pr[\mathsf{Enc}(K, n, x) = y] = \Pr[\mathsf{Enc}(K, n, x') = y]$$

Q.1 In this question only, we forget about the nonce N (i.e. $Y = \mathsf{Enc}(K, X)$). Recall the definition of perfect secrecy and prove that the above notion (without n) is equivalent when the support of X is $\mathbf{F}$.

HINT: Show that 1-time security implies $\forall x, x', y \quad \Pr[Y = y|X = x] = \Pr[Y = y|X = x']$.

We take an arbitrary distribution of X of support $S \subseteq \mathbf{F}$.
First of all, we observe that $\Pr[\mathsf{Enc}(K,x) = y] = \Pr[Y = y | X = x]$ *for* $x \in S$ *and any* y*. So, 1-time security implies*

$$\forall x, x' \in S \quad \forall y \in \mathbf{F} \quad \Pr[Y = y | X = x] = \Pr[Y = y | X = x']$$

Second, since $\Pr[Y = y] = \sum_{\xi \in S} \Pr[Y = y | X = \xi] \Pr[X = \xi]$*, we have that for all* $x \in S$ *and* $y \in \mathbf{F}$*,*

$$(\forall x' \in S \ \Pr[Y = y | X = x] = \Pr[Y = y | X = x']) \Longrightarrow \Pr[Y = y | X = x] = \Pr[Y = y]$$

Hence, 1-time security implies that for all $x \in S$ *and* $y \in \mathbf{F}$*, we have* $\Pr[Y = y | X = x] = \Pr[Y = y]$*. This is equivalent to the independence between* X *and* Y*, which is itself equivalent to perfect secrecy for the distribution of* X*. Therefore, 1-time security implies perfect secrecy for the distribution of* X*.*
To prove the contrary, let assume that we have perfect secrecy for a distribution of support $S = \mathbf{F}$*. For any* y*, we have*

$$\forall x \in \mathbf{F} \quad \Pr[Y = y | X = x] = \Pr[Y = y]$$

so

$$\forall x, x' \in \mathbf{F} \quad \Pr[Y = y | X = x] = \Pr[Y = y | X = x']$$

Which implies 1-time security.

Q.2 Propose an efficient nonce-based encryption system which is 1-time secure.

We can just copy the generalized Vernam cipher: $\mathsf{Enc}(K, n, x) = x + K$ *with* $K \in \mathbf{F}$ *uniformly distributed, where we just ignore the nonce. We know that it provides perfect secrecy in a group, and a field is a group for the addition. Using the previous question, we deduce that it is 1-time secure.*

Q.3 We say that the encryption system is 2-time secure if for any $x_1, x_2, x_1', x_2', y_1, y_2, n_1, n_2 \in \mathbf{F}$ such that $n_1 \neq n_2$, we have

$$\Pr[\mathsf{Enc}(K, n_1, x_1) = y_1, \mathsf{Enc}(K, n_2, x_2) = y_2] = \Pr[\mathsf{Enc}(K, n_1, x_1') = y_1, \mathsf{Enc}(K, n_2, x_2') = y_2]$$

Prove that 2-time security implies that the key space is at least as large as $\mathbf{F}^2$.
HINT: Use $\mathsf{Enc}'(K, (n_1, n_2), (x_1, x_2)) = (\mathsf{Enc}(K, n_1, x_1), \mathsf{Enc}(K, n_2, x_2))$.

The suggested Enc' *is another encryption system over the message domain* $\mathbf{F}^2$ *and the nonce space* $\{(n_1, n_2) \in \mathbf{F}^2 : n_1 \neq n_2\}$*. Clearly, the 2-time security of* Enc' *is equivalent to the 1-time security of* Enc' *which is itself equivalent to perfect secrecy. Due to the Shannon result, we deduce that the domain of* K *is at least as large as the message domain.*

Q.4 Propose a 2-time secure encryption system. (Prove that it is secure.)

We use $K \in \mathbf{F}^2$ and write $K = (\alpha, \beta)$. We define $\mathsf{Enc}((\alpha,\beta), n, x) = x + \alpha n + \beta$. Clearly, $p = \Pr[\mathsf{Enc}((\alpha,\beta), n_1, x_1) = y_1, \mathsf{Enc}((\alpha,\beta), n_2, x_2) = y_2] = \Pr[\alpha n_1 + \beta = y_1 - x_1, \alpha n_2 + \beta = y_2 - x_2]$. We can write it

$$p = \Pr\left[\begin{pmatrix} n_1 & 1 \\ n_2 & 1 \end{pmatrix}\begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} y_1 - x_1 \\ y_2 - x_2 \end{pmatrix}\right]$$

Since $n_1 \neq n_2$, the matrix is non-singular and we have

$$p = \Pr\left[\begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} n_1 & 1 \\ n_2 & 1 \end{pmatrix}^{-1}\begin{pmatrix} y_1 - x_1 \\ y_2 - x_2 \end{pmatrix}\right] = \frac{1}{\#\mathbf{F}^2}$$

This does not depend on $x_1, x_2, y_1, y_2, n_1, n_2$ and so it is the same for $x_1', x_2', y_1, y_2, n_1, n_2$. Therefore, the system is 2-time secure.

Q.5 Propose a definition for d-time security and propose a nonce-based system which achieves it.

We say that the encryption system is d-time secure if for any $x, x', y, n \in \mathbf{F}^d$ such that all coordinates of n are pairwise different, we have

$$\Pr\left[\bigwedge_i \mathsf{Enc}(K, n_i, x_i) = y_i\right] = \Pr\left[\bigwedge_i \mathsf{Enc}(K, n_i, x_i') = y_i\right] =$$

We can generalize the previous construction by writing $K \in \mathbf{F}^d$ as $K = (k_0, \ldots, k_{d-1})$ and $\mathsf{Enc}(K, n, x) = x + \sum_{i=0}^{d-1} k_i n^i$.
We could prove security the same way by obtaining the matrix

$$\begin{pmatrix} 1 & n_1 & n_1^2 & \cdots & n_1^{d-1} \\ 1 & n_2 & n_2^2 & \cdots & n_2^{d-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & n_d & n_d^2 & \cdots & n_d^{d-1} \end{pmatrix}$$

This is a Vandermonde matrix and it is well known that it is non-singular when the n_i are pairwise different.

2 Multiexponentiation

The following exercise is inspired from Algorithms for Multi-exponentiation *by Möller, published in proceedings of SAC 2001.*

In this exercise, we consider algorithms to compute g^e in an Abelian group G (which uses multiplicative notations). Given an integer e, we denote by $e[i] = \lfloor \frac{e}{2^i} \rfloor \bmod 2$ the ith bit of e (e.g. $e[0]$ is the least significant bit). We denote by $e[j \cdots i]$ the number obtained by concatenating the bits from the jth to the ith. (For instance, for $e = 23 = 10111_2$, we have $e[4 \cdots 0] = 10111_2 = 23$ and $e[3 \cdots 1] = 011_2 = 3$.) We recall the square-and-multiply algorithm from left to right.

$\mathsf{Exp}(g, e)$:

1: **for** $i = \ell - 1$ down to 0 **do**
2: **if** $i = \ell - 1$ **then**
3: $x \leftarrow 1$
4: **else**
5: $x \leftarrow x^2$
6: **end if**
7: **if** $e[i] = 1$ **then**
8: $x \leftarrow x \times g$
9: **end if**
10: **end for**
11: **return** x

Q.1 Assuming that e is a random string of ℓ bits, what are the average number of squarings and the average number of multiplications in the square-and-multiply algorithm?

In the algorithm, we have one squaring for $i = \ell - 2, \ldots, 0$, so $\ell - 1$ squarings in total. For $i = \ell - 1, \ldots, 0$, we have one multiplication if $e[i] = 1$ so with probability $\frac{1}{2}$. On average, we have $\ell/2$ multiplications.

Q.2 We now use the so-called 2^w-ary method: we first make a precomputation of all g^b for $b = 0, \ldots, 2^w - 1$, we split e in blocks of w bits, then we scan all the blocks from left to right.

Q.2a Fully specify the precomputation algorithm with input g.

We want to precompute $T\{b\} = g^b$ in a dictionary T. We could have made a loop of $2^w - 2$ multiplications by g. However, assuming that squaring is faster than multiplication, there is an advantage in computing the table entries using squarings and others by one multiplication by g.

$\mathsf{Precomp}(g)$*:*

1: $T\{1\} \leftarrow g$
2: ***for*** $b = 1$ *to* $2^{w-1} - 1$ ***do***
3: $T\{2b\} \leftarrow T\{b\}^2$
4: $T\{2b + 1\} \leftarrow T\{2b\} \times g$
5: ***end for***
6: ***return*** T

Q.2b What is the number of squarings and of multiplications?

The number of squarings is $\frac{1}{2}2^w - 1$. The number of multiplications is also $\frac{1}{2}2^w - 1$. The sum is $2^w - 2$ which is the same as we obtain when making a loop of $2^w - 2$ multiplications. However, having half of them becoming a squaring gives an advantage.

Q.2c Fully specify the algorithm to compute g^e using the results of the precomputation.

```
Exp(e, T):
 1: ℓ_w ← ⌈ℓ/w⌉
 2: for i = ℓ_w − 1 down to 0 do
 3:     if i = ℓ_w − 1 then
 4:         x ← 1
 5:     else
 6:         for j = 0 do to w − 1
 7:             x ← x^2
 8:         end for
 9:     end if
10:     b ← e[iw + w − 1 ··· iw]
11:     if b ≠ 0 then
12:         x ← x × T{b}
13:     end if
14: end for
15: return x
```

Q.2d What is the expected number of squarings and of multiplications in this phase?

The number of squarings is $w\lceil\frac{\ell}{w}\rceil - w$. The $b \neq 0$ event occurs with probability $1 - 2^{-w}$. Hence, the expected number of multiplications is $\lceil\frac{\ell}{w}\rceil(1 - 2^{-w})$.

Q.2e Assuming that ℓ is divisible by w, compute the sum of the total number of squarings and of multiplications in the two phases and compare with the normal square-and-multiply algorithm for $w = 1, 2, 3, 4$ and say for which ℓ which algorithm is better.

The total is

$$2^w - 2 + \ell - w + \frac{\ell}{w}(1 - 2^{-w})$$

With the square-and-multiply algorithm, it is $\frac{3}{2}\ell - 1$.
The 2^w-ary method with $w = 1$ is identical to the square-and-multiply algorithm. We compute the total for $w = 1, 2, 3, 4$ in the table below. The curves intersect sequentially and we obtain the following best algorithms.

ℓ :	0	8	36	122.2
best w:	1	2	3	4
total	$\frac{3}{2}\ell - 1$	$\frac{11}{8}\ell$	$\frac{31}{24}\ell + 3$	$\frac{79}{64}\ell + 10$

Q.3 We now want to compute $g_1^{e_1} \cdots g_k^{e_k}$, where all e_i are written with ℓ bits.

Q.3a What is the expected number of squarings and multiplications if we use the naïve method based on the square-and-multiply algorithm?

The naïve method will compute iteratively each $g_i^{e_i}$ and multiply them together. The total number of squarings is $k(\ell-1)$ and the expected number of multiplications is $k\ell/2$. In total, this sums to $\frac{3}{2}k\ell - k$.

Q.3b Inspired by the 2^w-ary method, propose an algorithm based on the precomputation of $T\{b_1,\ldots,b_k\} = g_1^{b_1}\cdots g_k^{b_k}$ for all the 2^{kw} possible combinations of blocks. (We assume that $g_1,\ldots,g_k$ are available for precomputation.)

```
Precomp(g_1, ..., g_k):
 1: S ← ∅
 2: for i = 1 to k do
 3:     b'_j ← 1_{j=i} for j = 1, ..., k
 4:     b' ← (b'_1, ..., b'_k)
 5:     T{b'} ← g_i
 6:     S ← S ∪ {b'}
 7: end for
 8: for c = 2 to k do
 9:     for each I ⊆ {1, ..., k} of size c do
10:         take i from I
11:         b''_j ← 2b_j + 1_{j∈I} for j = 1, ..., k
12:         b'' ← (b''_1, ..., b''_k)
13:         b'_j ← b''_j − 1_{j=i} for j = 1, ..., k
14:         b' ← (b'_1, ..., b'_k)
15:         T{b''} ← T{b'} × g_i
16:         S ← S ∪ {b''}
17:     end for
18: end for
19: for u = 2 to w do
20:     S' ← ∅
21:     for each b ∈ S do
22:         (b_1, ..., b_k) ← b
23:         T{(2b_1, ..., 2b_k)} ← T{b}^2
24:         S' ← S' ∪ {(2b_1, ..., 2b_k)}
25:         for c = 1 to k do
26:             for each I ⊆ {1, ..., k} of size c do
27:                 take i from I
28:                 b''_j ← 2b_j + 1_{j∈I} for j = 1, ..., k
29:                 b'' ← (b''_1, ..., b''_k)
30:                 b'_j ← b''_j − 1_{j=i} for j = 1, ..., k
31:                 b' ← (b'_1, ..., b'_k)
32:                 T{b''} ← T{b'} × g_i
33:                 S' ← S' ∪ {b''}
34:             end for
35:         end for
36:     end for
37:     S ← S'
38: end for
39: return T

Exp(e_1, ..., e_k, T):
40: ℓ_w ← ⌈ℓ/w⌉
41: for i = ℓ_w − 1 down to 0 do
42:     if i = ℓ_w − 1 then
43:         x ← 1
44:     else
45:         for j = 0 do to w − 1
46:             x ← x^2
47:         end for
48:     end if
49:     b_j ← e_j[iw + w − 1 ··· iw] for j = 1, ..., k
50:     b ← (b_1, ..., b_k)
51:     if b ≠ (0, ..., 0) then
52:         x ← x × T{b}
53:     end if
54: end for
55: return x
```

Q.3c What is the expected number of squarings and multiplications in each phase? Make a comparison with the naïve algorithm for $k = 2$ as it was done for the $k = 1$ case.

During precomputation, the number of squarings is $n_2 = (2^k - 1)(1 + 2^k + \cdots + 2^{kw-2k}) = 2^{k(w-1)} - 1$. *The number of multiplications is* $n_\times = 2^k - k - 1 + (2^k - 1)n_2 = 2^{kw} - 2^{k(w-1)} - k$. *During the second phase, the number of squarings is* $n'_2 = w(\ell_w - 1)$ *and the expected number of multiplications is* $n'_\times = \ell_w$.
If w *divides* ℓ, *we have* $\ell_w = \ell/w$ *and the total is*

$$n = 2^{k(w-1)} - 1 + 2^{kw} - 2^{k(w-1)} - k + w(\frac{\ell}{w} - 1) + \frac{\ell}{w} = 2^{kw} - k + \ell - w + \frac{\ell}{w} - 1$$

For $k = 2$, *this is* $n = 2^{2w} + \ell - w + \frac{\ell}{w} - 3$ *compared to* $3\ell - 2$ *for the naïve method. We obtain the following comparison.*

ℓ :	*0*	*2*	*22*	*282*	*2292*
best algo	*naïve*	$w = 1$	$w = 2$	$w = 3$	$w = 4$
total	$3\ell - 2$	2ℓ	$\frac{3}{2}\ell + 11$	$\frac{4}{3}\ell + 58$	$\frac{5}{4}\ell + 249$