Cryptography and Security — Final Exam
Solution

Serge Vaudenay

19.1.2024

- duration: 3h
- 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 The Security of Nonce-Based Symmetric Encryption

We recall the CPCA decryption game against non-based symmetric encryption on message space $\mathcal{D}$, nonce space $\mathcal{N}$, and key space $\{0,1\}^k$.

```
Game
 1: $K \xleftarrow{\$} \{0,1\}^k$
 2: $X_0 \xleftarrow{\$} \mathcal{D}$, $N_0 \xleftarrow{\$} \mathcal{N}$
 3: $\mathsf{Used} \leftarrow \{N_0\}$
 4: $Y_0 \leftarrow \mathsf{Enc}(K, N_0, X_0)$
 5: $\mathcal{A}^{\mathsf{OEnc},\mathsf{ODec}}(N_0, Y_0) \to X$
 6: return $1_{X=X_0}$

Oracle $\mathsf{OEnc}(N, X)$:
 7: if $N \in \mathsf{Used}$ then return $\bot$
 8: $\mathsf{Used} \leftarrow \mathsf{Used} \cup \{N\}$
 9: return $\mathsf{Enc}(K, N, X)$

Oracle $\mathsf{ODec}(N, Y)$:
10: if $(N, Y) = (N_0, Y_0)$ then return $\bot$
11: return $\mathsf{Dec}(K, N, Y)$
```

Q.1 List the differences with the CPA decryption game. Do the same for the CPCA key recovery game and the CPA key recovery game.

For CPA games, we should remove the ODec *oracle.*
For key recovery, lines 2, 4, and 10 are removed and there is no N_0 or X_0 to add in Used *nor to give to $\mathcal{A}$. The winning condition is that the output of $\mathcal{A}$ on line 5 is equal to K instead of X_0.*

Q.2 Recall how to adapt the Vernam cipher to make a nonce-based symmetric encryption scheme with a PRNG, and show that it is insecure against CPCA decryption attacks.

A Vernam-based cipher works as $\mathsf{Enc}(K,N,X) = X \oplus \mathsf{PRNG}(K,N)$*. We can define the following adversary:*

```
1: pick $Y$ as long as $Y_0$ but different
2: call $\mathsf{ODec}(N_0, Y) \to Z$
3: set $X = Z \oplus Y \oplus Y_0$
4: return $X$
```

Clearly, we have

$$X = Z \oplus Y \oplus Y_0 = \mathsf{PRNG}(K,N_0) \oplus Y_0 = X_0$$

so the adversary has advantage 1 with a single ODec *query.*

Q.3 We change the CPA decryption attack game by allowing nonce reuse in the encryption oracle. Show that a nonce-based symmetric encryption scheme based on the Vernam cipher is insecure in this game.

We only use one OEnc *oracle call with nonce* N_0*. This is normally illegal but this question of the exercise allowed it.*

```
1: pick $X'$ as long as $Y_0$
2: call $\mathsf{OEnc}(N_0, X') \to Y'$
3: set $X = Y' \oplus X' \oplus Y_0$
4: return $X$
```

We now have

$$X = Y' \oplus X' \oplus Y_0 = \mathsf{PRNG}(K,N_0) \oplus Y_0 = X_0$$

so the adversary has advantage 1 with a single OEnc *query with nonce misuse.*

Q.4 We use a block cipher C_K and assume that the block space is given a finite field structure $\mathbf{F} = \mathcal{D}$. We define $\mathsf{Enc}(K,N,\mathsf{pt}) = \mathsf{pt} \times K + C_K(N)$, for $K \in \mathbf{F}^*$ and $\mathsf{pt} \in \mathbf{F}$. Show that this nonce-based symmetric encryption system is insecure against CPCA key recovery attacks.

```
1: pick $Y_1, Y_2 \in \mathbf{F}$ different
2: pick $N \in \mathbf{F}^*$
3: call $\mathsf{ODec}(N, Y_1) \to X_1$
4: call $\mathsf{ODec}(N, Y_2) \to X_2$
5: set $K' = (Y_2 - Y_1)/(X_2 - X_1)$
6: return $K'$
```

Clearly, two plaintext/ciphertext pairs (X_1, Y_1) *and* $X_1, Y_2)$ *with same nonce* N *give two linear equations with unlnowns* K *and* $C_K(N)$*. It solves by* $K' = K$*.*

Q.5 For a stateless implementation of the message sender, what would you recommend for a nonce length in practice and why? (Here, stateless means that the sender does not keep more than the secret key in memory in between two messages to send.)

If the sender is stateless, it cannot keep a counter or any track of the used nonces. However, the nonce must not be reused. Hence, the only way is to use a random nonce which is long enough to avoid repetitions in practice. Due to the birthday paradox, B^2 *must be negligible compared to the nonce space size, where* B *is an upper bound on the number of messages the sender will ever send with the same key. By assuming* $B = 2^{40}$*, having a nonce of 128 bits should be safe. The probability of a collision after* B *samples is about* $1 - e^{B^2 2^{-128}/2} \approx 2^{-49}$.

2 Privacy Pass

We consider a group G of prime order q. We use additive notations for this group. Let H be a random function from the set of bistrings to $G - \{0\}$. We assume that H is publicly known. We consider a function family $(f_K)_{K \in \mathbf{Z}_q^*}$ from the set of bitstrings to G defined by $f_K(x) = K \cdot H(x)$. We call K a secret key. We design a token-issuance protocol between a client and a server as follows:

– The client has a bitstring x as input. The server has a key K as input.
– The client picks $\lambda \in \mathbf{Z}_q^*$, sets $X = \lambda \cdot H(x)$, and sends X to the server.
– The server computes $Y = K \cdot X$ and returns it to the client.
– The client computes $Z = \frac{1}{\lambda} \cdot Y$ and takes it as an output.

A token is a (x, Z) pair. After the token is issued by the above protocol, the client can redeem the token to the server. The server accepts the token as valid if it satisfies $Z = f_K(x)$. We consider the following security game:

```
Game $\Gamma_b$
  1: pick $H : \{0,1\}^* \to G - \{0\}$
  2: $K \xleftarrow{\$} \mathbf{Z}_q^*$
  3: pick $F : \{0,1\}^* \to G - \{0\}$
  4: $\mathcal{A}^{\mathsf{OH},\mathsf{OF}} \to z$
  5: return $z$

Oracle $\mathsf{OH}(x)$:
  6: return $H(x)$

Oracle $\mathsf{OF}(x)$:
  7: if $b = 0$ then return $F(x)$
  8: return $f_K(x)$
```

The advantage of adversary $\mathcal{A}$ is $\mathsf{Adv} = \Pr[\Gamma_1 \to 1] - \Pr[\Gamma_0 \to 1]$. We assume that this construction is secure in the sense that for any adversary $\mathcal{A}$ limited to a *feasible* complexity and number of oracle queries, the advantage is *negligible*.

Q.1 If we remove the first line of the game and the oracle OH, what is this security notion? Why did we add the oracle OH?

This is the pseudorandom function (PRF) security notion where we added access to the random function H through oracle OH. (This is called PRF in the random oracle model.) We added access to H because this function is supposed to be publicly known, so accessible by the adversary. We cannot provide the table of H which is too large (actually, it has infinite size) so we rather give access to an oracle which evaluate H.

Q.2 Let x be a random bitstring following an arbitrary distribution, H be an arbitrary function (fixed), K be the random key, and λ be the random mask. In the issuance protocol, prove that X is independent from x.

Let g be a generator of G and $L = \log_g H(x)$. Clearly, λ and L are independent, and in $\mathbf{Z}_q^*$. We have $\log_g X = \lambda L$. Since λ is uniform in $\mathbf{Z}_q^*$ and independent from L, then $\log_g X$ is independent from x. (This is a result from Chapter 1 as the group operation of two independent random variables, one being uniform in the group.) Thus, X is independent from x. It reveals no information about x.

Q.3 The token-issuance protocol is sometimes called an *oblivious* evaluation protocol for $f_K(x)$. Explain (guess) why?

This is because nothing about x leaks to the server during issuance. If the server engages with several issuance sessions with several clients and is later on given one token for redeem, the server cannot figure out to which issuance session is corresponds.

Q.4 Assume that a malicious client can interract with a token-issuance-server oracle and with a token-redeem-server oracle. We want to formalize the security notion saying that if the adversary interacts n times with the issuance server oracle, then it cannot produce $n+1$ pairwise different valid tokens. This is called one-more unforgeability (OMUF). Propose a game to define OMUF security.

Here is the OMUF game:

```
Game
1:  $K \xleftarrow{\$} \mathbf{Z}_q^*$
2:  $n \leftarrow 0$
3:  $\mathcal{A}^{\mathsf{OIss},\mathsf{ORed}} \to (x_1, Z_1), \ldots, (x_m, Z_m)$
4:  if $m \neq n+1$ then return 0
5:  for $i = 1$ to $m$ do
6:      for $j = i+1$ to $m$ do
7:          if $x_i = x_j$ then return 0
8:      end for
9:      $\mathsf{ORedeem}(x_i, Z_i) \to b_i$
10:     if $b_1 = 0$ then return 0
11: end for
12: return 1

Oracle $\mathsf{OIss}(X)$:
13: $n \leftarrow n+1$
14: return $K \cdot X$

Oracle $\mathsf{ORed}(x, Z)$:
15: return $1_{Z=K\cdot H(x)}$
```

The two oracles are OIss *and* ORed *which model the interactions with the server in issuance or redeem. We added a counter n to count the number of calls to the issuance server oracle* OIss. *Eventually, the adversary produces m tokens. The winning condition is that $m = n+1$, that all x_i are pairwise different, and that all tokens are valid.*

3 Discrete Log in a Small Set

We consider a group G of large prime order q with a generator g. Given $y \in G$, the discrete logarithm problem is the problem of finding $x \in \mathbf{Z}_q$ such that $y = x \cdot g$. One generic algorithm to solve that is the Baby-Step Giant-Step algorithm which is recalled below.

```
Input: $g$ and $y$ in a group $G$, $B$ an upper bound for $\#G$
Precomputation ($y$ not provided)
 1: set $\ell = \lceil\sqrt{B}\rceil$
 2: for $i = 0, \ldots, \ell - 1$ do
 3:     define $T\{(i\ell) \cdot g\} = i$
 4: end for
Algorithm ($y$ provided)
 5: for $j = 0, \ldots, \ell - 1$ do
 6:     compute $z = y - j \cdot g$
 7:     if $T\{z\}$ is defined then
 8:         yield $x = \ell T\{z\} + j$ and stop
 9:     end if
10: end for
```

We let S be a small subset of $\mathbf{Z}_q$ consisting of all integers between a and b, with $b - a$ small. That is, $S = \{a, a+1, \ldots, b\}$. We call S a small interval of $\mathbf{Z}_q$. Given $y \in G$, we consider the problem of finding $x \in \mathbf{Z}_q$ such that $y = x \cdot g$, when we know that there exists one such x in S.

Q.1 Optimize the implementation of the Baby-Step Giant-Step in order to minimize the total number of group additions. Analyze the precomputation complexity (total number of group additions in precomputation), time complexity (total number of group additions in the algorithm using y), and memory complexity (total number of entries defined in T). (Give the worst case and the average case.)

To optimize, we have to clarify what is the notion of step. *In the precomputation, we can compute $(i\ell) \cdot g$ after we computed $((i-1)\ell) \cdot g$ by just adding $\ell \cdot g$. Adding this is the notion of making a* giant step *and it costs one addition, assuming that $\ell \cdot g$ was computed before the loop. After the precomputation phase, we can compute $-j \cdot g$ after we computed $-(j-1) \cdot g$ by adding $-g$. This is a small step and it requires to precompute $-g$. We obtain*

```
Input: $g$, $y$, $G$, $B$
Precomputation ($y$ not provided)
 1: set $\ell = \lceil \sqrt{B} \rceil$
 2: compute $S = \ell \cdot g$
 3: set $z = 0$
 4: for $i = 0, \ldots, \ell - 1$ do
 5:     define $T\{z\} = i$
 6:     set $z \leftarrow z + S$
 7: end for
Algorithm ($y$ provided)
 8: compute $s = -g$
 9: set $z = y$
10: for $j = 0, \ldots, \ell - 1$ do
11:     if $T\{z\}$ is defined then
12:         yield $x = \ell T\{z\} + j$ and stop
13:     end if
14:     set $z \leftarrow z + s$
15: end for
```

Assuming that $\ell \cdot g$ up to $2 \log_2 \sqrt{B}$ additions using the double-and-square algorithm and that $-g$ has the same cost as one addition, the precomputation needs at most $\sqrt{B} + \log_2 B$ additions (worst case and average case are the same), the discrete log phase takes at most $\sqrt{B}$ additions (half of it on average), and the T contains $\sqrt{B}$ entries.

Q.2 Propose a variant of this algorithm for the problem in this exercise (with solutions in S) and analyze its precomputation complexity, time complexity, and memory complexity.

We cut the S interval to make baby steps and giant steps.

```
Input: $g$, $y$, $G$, $a$, $b$
Precomputation (y not provided)
 1: set $\ell = \lceil \sqrt{b-a} \rceil$
 2: compute $S = \ell \cdot g$
 3: compute $z = a \cdot g$
 4: for $i = 0, \ldots, \ell - 1$ do
 5:     define $T\{z\} = i$
 6:     set $z \leftarrow z + S$
 7: end for
Algorithm (y provided)
 8: compute $s = -g$
 9: set $z = y$
10: for $j = 0, \ldots, \ell - 1$ do
11:     if $T\{z\}$ is defined then
12:         yield $x = \ell T\{z\} + j$ and stop
13:     end if
14:     set $z \leftarrow z + s$
15: end for
```

The complexity analysis is obtained by replacing B by $b-a$. We shall as the cost of computing the starting point $a \cdot g$ of the precomputation. The precomputation needs at most $\sqrt{b-a} + 2\log_2(b-a)$ additions (worst case and average case are the same), the discrete log phase takes at most $\sqrt{b-a}$ additions (half of it on average), and the T contains $\sqrt{b-a}$ entries.

Q.3 What time-memory tradeoffs are possible in the algorithm?

We do not have to cut the interval in $\sqrt{b-a}$ giant steps and the same number of baby steps in between. If we cut it in M giant steps, with $\ell = \lceil \frac{b-a}{M} \rceil$, we obtain M entries in T, same precomputation complexity (by neglecting the S and z precomputations), and $\frac{b-a}{M}$ baby steps in the discrete log phase. The tradeoff is thus M for precomputation and memory complexity versus $\frac{b-a}{M}$ time complexity in the discrete log phase.

Q.4 Propose a way to adapt the ElGamal encryption to encrypt a *small* bitstring instead of a group element. Specify the size of this plaintext.

Recall that the ElGamal cryptosystem can only encrypt group elements. Taking the bistring pt *as an integer written in binary, we can use ElGamal to encrypt* $\mathsf{pt} \cdot g$*, which is a group element. After ElGamal decryption, we can compute the discrete logarithm to recover* pt*. For plaintexts of 64 bits, the above algorithms work with complexity 2^{32} and a precomputed table of 2^{32} entries, which is doable. It is already expensive, compared to alternate cryptosystems.*