Advanced Cryptography — Midterm Exam
Solution

Serge Vaudenay

14.4.2022

- duration: 1h45
- any document allowed
- 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

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

1 2-Move Authenticated Key Agreement

The traditional Diffie-Hellman key agreement scheme is a 2-move protocol. The interface can be modeled in the following way:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$ sets up public parameters pp using a security parameter λ.
- $\mathsf{Send}(\mathsf{pp}) \to (\mathsf{esk}, \mathsf{epk})$ generates an ephemeral key pair where epk is to be sent to the counterpart.
- $\mathsf{Receive}(\mathsf{pp}, \mathsf{esk}, \mathsf{epk}) \to K$ receives the counterpart's epk and generates the shared key K.

Both participants are supposed to send and receive to derive their local K. The protocol is meant to resist to passive attacks with the notion of key indistinguishability. We extend this primitive in order to authenticate the final key by using a long-term key. For this, we add the following algorithm in the interface:

- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{lsk}, \mathsf{lpk})$ generates a long term key pair for a user, where lpk is publicly associated to the user and lsk is kept secret.

In addition, Send takes as input the long-term secret of the user and $\mathsf{Receive}$ takes as input the long-term public key of the counterpart.

Q.1 Rewrite the entire interface and define the correctness notion using a fully specified game.

The interface is now:

- $\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{pp}$
- $\mathsf{KeyGen}(\mathsf{pp}) \rightarrow (\mathsf{lsk}, \mathsf{lpk})$
- $\mathsf{Send}(\mathsf{pp}, \mathsf{lsk}) \rightarrow (\mathsf{esk}, \mathsf{epk})$
- $\mathsf{Receive}(\mathsf{pp}, \mathsf{esk}, \mathsf{lpk}, \mathsf{epk}) \rightarrow K/\bot$

Correctness is defined by having the following game always returning 1:

$\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{pp}$
$\mathsf{KeyGen}(\mathsf{pp}) \rightarrow (\mathsf{lsk}_A, \mathsf{lpk}_A)$ ▷ *initialize Alice*
$\mathsf{KeyGen}(\mathsf{pp}) \rightarrow (\mathsf{lsk}_B, \mathsf{lpk}_B)$ ▷ *initialize Bob*
$\mathsf{Send}(\mathsf{pp}, \mathsf{lsk}_A) \rightarrow (\mathsf{esk}_A, \mathsf{epk}_A)$ ▷ *Alice sends*
$\mathsf{Send}(\mathsf{pp}, \mathsf{lsk}_B) \rightarrow (\mathsf{esk}_B, \mathsf{epk}_B)$ ▷ *Bob sends*
$K_A \leftarrow \mathsf{Receive}(\mathsf{pp}, \mathsf{esk}_A, \mathsf{lpk}_B, \mathsf{epk}_B)$ ▷ *Alice receives*
$K_B \leftarrow \mathsf{Receive}(\mathsf{pp}, \mathsf{esk}_B, \mathsf{lpk}_A, \mathsf{epk}_A)$ ▷ *Bob receives*
return $1_{K_A = K_B \neq \bot}$

A recurrent mistake is to describe a game with an adversary $\mathcal{A}$. It is possible to do so must it is most likely to be incorrect. First of all, the adversary must be quantified. We cannot say "the scheme is correct if there exists an $\mathcal{A}$ such that..." because the guaranty of the existence of $\mathcal{A}$ may not match what we think is the proper usage of the scheme. We cannot say "the scheme is correct if for all $\mathcal{A}$ we have..." because the adversary doing nothing is unlikely to make the final outcome correct. The adversary in correctness may be needed when there are many possible ways to use the scheme, with choices which could be adversarialy made. This is not the case here. The protocol follows some well identified sequence: setup the keys, generate the ephemeral ones, exchange them, and complete. There is no place for choices by an adversary here.

To model security against active attacks, we can no longer assume that the protocol is honestly executed and give the transcript to the adversary. Instead, we use oracles to model honest Alice honest Bob running Send and $\mathsf{Receive}$. These oracles shall allow multiple concurrent sessions. Hence, we consider the game in Fig. 1.

The instruction **ensure** tests if the following predicate is true and causes the oracle to return $\bot$ if it is not the case.

Q.2 Fully define the key indistinguishability notion based on this game.

Motivate why $\mathsf{OReceive}$ returns whether $K_P \neq \bot$.

Explain why OTest ensures $K_1 \neq \bot$.

Game Γ_b:
initialize state to empty
$\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$
$\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{lsk}_A, \mathsf{lpk}_A)$
$\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{lsk}_B, \mathsf{lpk}_B)$
$\mathcal{A}^{\mathsf{oracles}}(\mathsf{pp}, \mathsf{lpk}_A, \mathsf{lpk}_B) \to z$
return z

Oracle $\mathsf{OReceive}(P, \mathsf{sid}, \mathsf{lpk}, \mathsf{epk})$:
ensure $P \in \{A, B\}$
ensure $\mathsf{state}[P, \mathsf{sid}]$ exists with only two elements
$\mathsf{state}[P, \mathsf{sid}] \to (\mathsf{esk}_P, \mathsf{epk}_P)$
$K_P \leftarrow \mathsf{Receive}(\mathsf{pp}, \mathsf{esk}_P, \mathsf{lpk}, \mathsf{epk})$
select K_0 at random
$\mathsf{state}[P, \mathsf{sid}] \leftarrow (\mathsf{esk}_P, \mathsf{epk}_P, \mathsf{lpk}, \mathsf{epk}, K_0, K_P)$
return $1_{K_P \neq \perp}$

Oracle $\mathsf{OSend}(P, \mathsf{sid})$:
ensure $P \in \{A, B\}$
ensure $\mathsf{state}[P, \mathsf{sid}]$ does not exist
$\mathsf{Send}(\mathsf{pp}, \mathsf{lsk}_P) \to (\mathsf{esk}_P, \mathsf{epk}_P)$
$\mathsf{state}[P, \mathsf{sid}] \leftarrow (\mathsf{esk}_P, \mathsf{epk}_P)$
return epk_P

Oracle $\mathsf{OTest}(P, \mathsf{sid})$:
ensure $P \in \{A, B\}$
ensure $\mathsf{state}[P, \mathsf{sid}]$ exists with six elements
$\mathsf{state}[P, \mathsf{sid}] \to (\mathsf{esk}_P, \mathsf{epk}_P, \mathsf{lpk}, \mathsf{epk}, K_0, K_1)$
ensure $K_1 \neq \perp$
return K_b

Fig. 1. Key indistinguishability game

The protocol is secure if for any PPT adversary $\mathcal{A}$, the advantage defined by

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr[\Gamma_1 \to 1] - \Pr[\Gamma_0 \to 1]$$

is a negligible function of λ.
The reason why $\mathsf{OReceive}$ *returns whether $K_P \neq \perp$ is because in real applications, the adversary will be able to figure out if a participant aborts or continues to interact after key agreement is over.*
If the adversary makes sure that a key agreement fails by having $\mathsf{Receive}$ *returning $\perp$,* OTest *should always return $\perp$ no matter the value of b. Otherwise, it is trivial to deduce b and break the security notion. This is why it ensures $K_1 \neq \perp$.*

Q.3 By using an adversary who makes Alice and Bob honestly execute the protocol, prove that security in the sense of the above game can easily be broken.

Propose a way to fix the game to get a sound security notion.

We use the following adversary:

Adversary $\mathcal{A}(\mathsf{pp}, \mathsf{lpk}_A, \mathsf{lpk}_B)$*:*

$\mathsf{OSend}(A, 1) \to \mathsf{epk}_A$
$\mathsf{OSend}(B, 1) \to \mathsf{epk}_B$
$\mathsf{OReceive}(A, 1, \mathsf{lpk}_B, \mathsf{epk}_B)$
$\mathsf{OReceive}(B, 1, \mathsf{lpk}_A, \mathsf{epk}_A)$
$\mathsf{OTest}(A, 1) \to K_A$
$\mathsf{OTest}(B, 1) \to K_B$
return $1_{K_A = K_B}$

If $b = 1$*, correctness implies that* $\mathcal{A}$ *always returns 1. If* $b = 0$*,* K_A *and* K_B *are set to independent random keys so are equal with negligible probability. Hence, the advantage is* $1 - \mathsf{negl}(\lambda)$*.*
One easy way to fix is to make sure that OTest *is used only once:*

Game Γ_b*:*

initialize tested *to* false
$\vdots$

Oracle $\mathsf{OTest}(P, \mathsf{sid})$*:*

ensure $\neg\mathsf{tested}$
ensure $P \in \{A, B\}$
ensure $\mathsf{state}[P, \mathsf{sid}]$ *exists with six elements*
$\mathsf{state}[P, \mathsf{sid}] \to (\mathsf{esk}_P, \mathsf{epk}_P, \mathsf{lpk}, \mathsf{epk}, K_0, K_1)$
ensure $K_1 \neq \bot$
$\mathsf{tested} \leftarrow \mathsf{true}$
return K_b

Another way is to make sure that K_0 *is selected the same on both ends when it should be the case. The big problem is to identify well when this should be the case. The adversary may call the two participants with different* sid*. Essentially, we need to check if a session* sid *for* A *is "partner" of a session* sid' *for* B*. This can be done as follows:*

Oracle $\mathsf{OReveal}(P, \mathsf{sid}, \mathsf{epk})$*:*

ensure $P \in \{A, B\}$
set Q *such that* $\{P, Q\} = \{A, B\}$
ensure $\mathsf{state}[P, \mathsf{sid}]$ *exists with only two elements*
$\mathsf{state}[P, \mathsf{sid}] \to (\mathsf{esk}_P, \mathsf{epk}_P)$
$K_P \leftarrow \mathsf{Receive}(\mathsf{pp}, \mathsf{esk}_P, \mathsf{lpk}_Q, \mathsf{epk})$
select K_0 *at random*
for *each* sid' *such that* $\mathsf{state}[Q, \mathsf{sid}']$ *exists with six elements* ***do***
    $\mathsf{state}[Q, \mathsf{sid}'] \to (\mathsf{esk}'_Q, \mathsf{epk}'_Q, \mathsf{lpk}'_P, \mathsf{epk}'_P, K'_0, K'_1)$
    if $(\mathsf{epk}_P, \mathsf{epk}, K_P) = (\mathsf{epk}'_P, \mathsf{epk}'_Q, K'_1)$ *then* $K_0 \leftarrow K'_0$
end for

There was an error in the specification of $\mathsf{OReceive}$ *in the exercise which created another security trouble. A few students have found it instead of the above problem. The mistake was to let* lpk *be an input to* $\mathsf{OReceive}$ *which could be maliciously selected by the adversary. As a consequence, the adversary could generate its own key pair, do a normal key agreement with one participant, test the key of that participant and compare with the key obtained by the adversary. It is another trivial attack. Instead,* $\mathsf{OReceive}$ *should not make* lpk *an input but rather use* lpk_Q *generated by the game (as in the above pseudocode).*

Q.4 Propose a protocol. Note: we do not require a security proof. The grade for this question will depend on the security of the proposed protocol.

We use a normal key agreement KA *(for instance the Diffie-Hellman protocol) and a digital signature scheme* DS*. Essentially, we sign the ephemeral public keys.*

$\mathsf{Setup}(1^\lambda)$:

- $\mathsf{KA.Setup}(1^\lambda) \to \mathsf{pp}_1$
- $\mathsf{DS.Setup}(1^\lambda) \to \mathsf{pp}_2$
- $\mathsf{pp} \leftarrow (\mathsf{pp}_1, \mathsf{pp}_2)$
- ***return*** pp

$\mathsf{KeyGen}(\mathsf{pp})$:

- $\mathsf{pp} \to (\mathsf{pp}_1, \mathsf{pp}_2)$
- $\mathsf{DS.KeyGen}(\mathsf{pp}_2) \to (\mathsf{lsk}, \mathsf{lpk})$
- ***return*** $(\mathsf{lsk}, \mathsf{lpk})$

$\mathsf{Send}(\mathsf{pp}, \mathsf{lsk})$:

- $\mathsf{pp} \to (\mathsf{pp}_1, \mathsf{pp}_2)$
- $\mathsf{KA.Send}(\mathsf{pp}_1) \to (\mathsf{esk}, \mathsf{epk}_0)$
- $\mathsf{DS.Sign}(\mathsf{pp}_2, \mathsf{lsk}, \mathsf{epk}_0) \to \sigma$
- $\mathsf{epk} \leftarrow (\mathsf{epk}_0, \sigma)$
- ***return*** $(\mathsf{esk}, \mathsf{epk})$

$\mathsf{Receive}(\mathsf{pp}, \mathsf{esk}, \mathsf{lpk}, \mathsf{epk})$:

- $\mathsf{pp} \to (\mathsf{pp}_1, \mathsf{pp}_2)$
- $\mathsf{epk} \leftarrow (\mathsf{epk}_0, \sigma)$
- $\mathsf{KA.Receive}(\mathsf{pp}_1, \mathsf{esk}, \mathsf{epk}_0) \to K$
- ***if*** $\neg\mathsf{DS.Verify}(\mathsf{pp}_2, \mathsf{lpk}, \mathsf{epk}_0, \sigma)$ ***then*** $K \leftarrow \perp$
- ***return*** K

2 Redundant-RSA Decryption

Let n be an RSA modulus of unknown factorization. We know that given $(x+1)^3 \bmod n$ and $x^3 \bmod n$ we can easily compute $x \bmod n$.

Q.1 Given n, $a = (x+1)^5 \bmod n$, and $b = x^3 \bmod n$, show how to compute $x \bmod n$ efficiently. Hint: x is a root of any polynomial which is a combination of $(z+1)^5 - a$ and $z^3 - b$ in $\mathbf{Z}_n$.

From a mathematical viewpoint, we consider the ideal of polynomials in $\mathbf{Z}_n[z]$ generated by $(z+1)^5-a$ and z^3-b. All polynomials in this ideal have x as a root. If we find a polynomial of degree 1, we can solve it and find x. We essentially compute the gcd of the two polynomials by using the Euclid algorithm.
We write equations in $\mathbf{Z}_n$ and we omit n for more readability. Since $x^3 = b$, we have

$$\begin{aligned} 0 &= (x+1)^5 - a \\ &= x^5 + 5x^4 + 10x^3 + 10x^2 + 5x + 1 - a \\ &= (b+10)x^2 + 5(b+1)x + 1 - a + 10b \end{aligned}$$

Hence

$$\begin{aligned} 0 &= (b+10)(x^3 - b) \\ &= x(-5(b+1)x - 1 + a - 10b) - b(b+10) \\ &= -5(b+1)x^2 - (1-a+10b)x - b(b+10) \end{aligned}$$

By making a linear combination of the two equations to make the x^2 terms cancel, we obtain

$$\begin{aligned} 0 &= 25(b+1)^2x + 5(b+1)(1-a+10b) - (b+10)(1-a+10b)x - b(b+10)^2 \\ &= (25(b+1)^2 - (b+10)(1-a+10b))x + 5(b+1)(1-a+10b) - b(b+10)^2 \end{aligned}$$

from which we deduce

$$x = \frac{5(b+1)(1-a+10b) - b(b+10)^2}{-25(b+1)^2 + (b+10)(1-a+10b)} \bmod n$$

which we can easily compute.