

UNIVERSITY
OF OSLO

TEK4500 – Introduction to Cryptography

Problem Set 9

Fall 2026

Problem 1

Read Chapter 14 in [Ros].

Problem 2

A *key encapsulation mechanism* (KEM) can be thought of as a specialized public-key encryption scheme, that only allows symmetric keys to be encrypted rather than arbitrary messages. More formally, a KEM is a tuple of algorithms $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$, where

- KeyGen is a probabilistic *key generation* algorithm that outputs a key-pair (dk, ek), where dk is the private *decapsulation* key and ek is the public *decapsulation* key.
- $\mathsf{Encaps} : \mathcal{PK} \to \mathcal{K} \times \mathcal{C}$ is a probabilistic *encapsulation* algorithm that, on input a public encapsulation key $ek \in \mathcal{PK}$, outputs a *symmetric* key $K \in \mathcal{K}$ and a ciphertext $C \in \mathcal{C}$. That is, $(K, C) \leftarrow \mathsf{Encaps}(ek)$. Notice that Encaps does not take a message as input!
- $\mathsf{Decaps} : \mathcal{SK} \times \mathcal{C} \to \mathcal{K}$ is a deterministic *decapsulation* algorithm that, on input a private decapsulation key $dk \in \mathcal{SK}$ and a ciphertext $C \in \mathcal{C}$, outputs a symmetric key $K \in \mathcal{K}$. That is, $K \leftarrow \mathsf{Decaps}(dk, C)$.

When using a KEM, Alice would use the *public* encapsulation key ek of Bob to create a symmetric key K and a corresponding ciphertext C. The symmetric key Alice would keep for herself, while the ciphertext C she would send to Bob. On receiving C, Bob would use his *private* decapsulation key dk to decapsulate C to obtain K.

a) Formulate a *correctness* requirement for a KEM.

b) Show how you can create a KEM scheme $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ from a public-key *encryption* scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, where

- $\Sigma.\mathsf{Enc} : \mathcal{PK} \times \{0,1\}^n \to \mathcal{C}$
- $\Sigma.\mathsf{Dec} : \mathcal{SK} \times \mathcal{C} \to \{0,1\}^n$

Note that the message space of Σ is $\{0,1\}^n$ for some fixed n. Include in your answer the corresponding key space and ciphertext space for your KEM. Are there any requirements on n in order for KEM to be secure?

c) Show how you can create a public-key *encryption* scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ from a KEM scheme $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$. What is the key space, message space, and ciphertext space of Σ?

Hint: Make use of a *symmetric* encryption scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ as an additional ingredient.

d) The Diffie-Hellman key exchange can also be used to create a KEM (let's call this DH-KEM for future reference). Specifically, in DH-KEM, the KeyGen algorithm would consist of picking a random exponent $b \leftarrow \{1, \ldots, |\mathbf{G}|\}$, and defining $dk = b$ and $ek = g^b$, where g is the generator for the cyclic group $\mathbf{G}$. What are the corresponding DH-KEM .Encaps and DH-KEM .Decaps algorithms?

Hint: Imagine $(dk, ek) = (b, g^b)$ as being the KEM key of Bob that he reuses across multiple connections. What would Alice do when trying to "encapsulate" a key towards Bob using ek?

e) The Diffie-Hellman protocol can be protected from man-in-the-middle attacks by signing the messages with a digital signature scheme. This makes the Diffie-Hellman protocol an *authenticated* key exchange protocol. Note that the signature keys are *long-term*, meaning that they are reused across multiple Diffie-Hellman exchanges.

It is also possible to create an authenticated key exchange protocol requiring no signatures at all, using solely a KEM scheme, This is shown in the figure below. Here, both Alice and Bob have their own long-term KEM key pairs, (dk_A, ek_A) and (dk_B, ek_B), respectively.

In the protocol above, Alice first creates a ciphertext C_B by using Bob's long-term encapsulation key ek_B. She also creates a one-time KEM key pair (dk, ek) which she sends to Bob together with C_B. Bob creates two ciphertexts C_A and C, using, respectively Alice's long-term encapsulation key ek_A, and the one-time key ek. Bob sends both C_A and C to Alice, and additionally decapsulates the ciphertext C_B using his long-term decapusulation key dk_B. When receiving C_A and C, Alice decapsulate both, using her long-term decapsulation key dk_A and the one-time key dk, respectively. Finally, both Alice and Bob derive a common shared secret by hashing K_A, K_B and K with some hash function H.

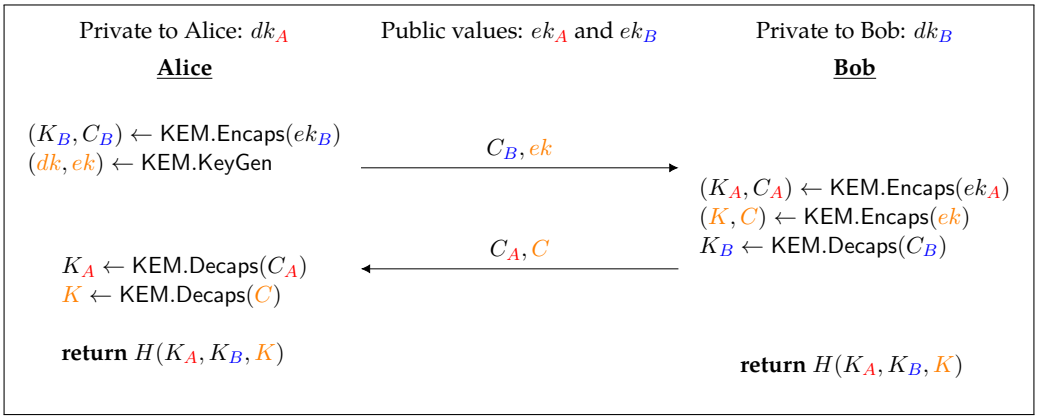


Figure 1: Authenticated key exchange from a KEM.

- Explain why an adversary (who doesn't know dk_A and dk_B) cannot perform a man-in-the-middle attack against the protocol above.
- What is the purpose of the one-time KEM in the protocol above? That is, what is the purpose of (dk, ek), C, and K?

ElGamal.KeyGen:

1: $x \twoheadleftarrow \{0, 1, \ldots, |\mathbf{G}| - 1\}$
2: $X \leftarrow g^x$
3: **return** $(sk = x, pk = X)$

ElGamal.Enc$(pk = X, M)$:

1: $y \twoheadleftarrow \{0, 1, \ldots |\mathbf{G}| - 1\}$
2: $Y \leftarrow g^y$
3: $Z \leftarrow X^y$
4: $C \leftarrow Z \circ M$
5: **return** (Y, C)

Figure 2: The Textbook ElGamal encryption scheme. It is parameterized by a cyclic group $\mathbf{G} = \langle g \rangle$.

Problem 3

The way we defined ElGamal in class is slightly different from how it is presented in most textbooks, which typically do not include the symmetric encryption scheme Σ. Instead, they encrypt the message directly using the derived key Z in a "one-time-pad" like fashion (see Fig. 2).

a) What is the message space for Textbook ElGamal?

b) Describe the decryption algorithm for Textbook ElGamal.

c) Show that Textbook ElGamal does not achieve IND-CCA security (Fig. 3).

$\mathbf{Exp}^{\text{ind-cca}}_{\Sigma}(\mathcal{A})$:

1: $b \leftarrow \{0, 1\}$
2: Ciphertexts $\leftarrow [\,]$
3: $(sk, pk) \twoheadleftarrow \Sigma$.KeyGen
4: $b' \leftarrow \mathcal{A}^{\mathcal{E}(\cdot), \mathcal{D}(\cdot)}(pk)$
5: **return** $b' \stackrel{?}{=} b$

$\mathcal{E}(M)$:

1: $C_0 \leftarrow \Sigma.\text{Enc}(pk, M)$
2: $C_1 \twoheadleftarrow \mathcal{C}(|M|)$
3: Ciphertexts.add(C_b)
4: **return** C_b

$\mathcal{D}(C)$:

1: **if** $C \in$ Ciphertexts **:** // Cheating!
2: **return** $\perp$
3: **return** $\Sigma.\text{Dec}(sk, C)$

$$\mathbf{Adv}^{\text{ind-cca}}_{\Sigma}(\mathcal{A})) = \left| 2 \cdot \Pr[\mathbf{Exp}^{\text{ind-cca}}_{\Sigma}(\mathcal{A}) \Rightarrow \text{true}] - 1 \right|$$

Figure 3: IND-CCA security experiment for a public-key encryption scheme Σ.

Problem 4

One way of upgrading an IND-CPA secure public-key encryption scheme Σ^{asym} into an IND-CCA secure one is to apply something called the Fujisaki-Okamoto (FO) tranformation. The FO-transform consists of essentially three steps:

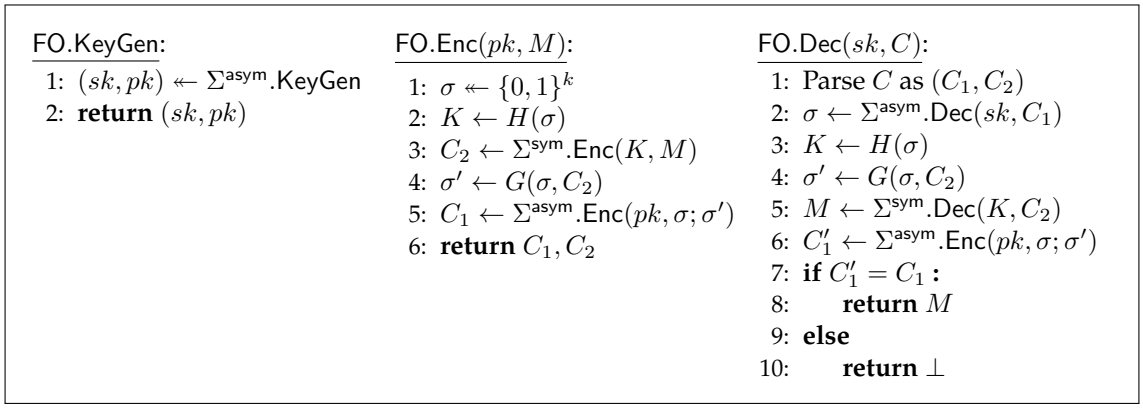


Figure 4: The FO-transform. It is parameterized by a public-key encryption scheme Σ^{asym}, a symmetric encryption scheme Σ^{sym}, and two hash functions H, G.

1. Generate a random bitstring σ. From σ derive a symmetric key K by hashing it with H, i.e. $K \leftarrow H(\sigma)$. With K encrypt the actual message M using a symmetric encryption scheme Σ^{sym}, yielding a ciphertext C_2.

2. Encrypt σ with the IND-CPA secure public-key encryption scheme Σ^{asym}, giving a ciphertext C_1. However, there's a twist to this encryption step. Normally, a public-key encryption algorithm generates its own internal randomness when encrypting a message, but here we feed in the random coins externally. Moreover, these random coins σ' are derived from σ and C_2 using another hash function G, i.e. $\sigma' \leftarrow G(\sigma, C_2)$.

   In particular, when encrypting σ we use σ' as the "internal" randomness of $\Sigma^{\mathsf{asym}}.\mathsf{Enc}$. To make this explicit we use the notation $C_2 \leftarrow \Sigma^{\mathsf{asym}}.\mathsf{Enc}_{pk}(\sigma; \sigma')$, as opposed to the usual notation $C_2 \leftarrow \Sigma^{\mathsf{asym}}.\mathsf{Enc}_{pk}(\sigma)$ where the internal randomess is "hidden". Thus, $\Sigma^{\mathsf{asym}}.\mathsf{Enc}_{pk}(\sigma)$ is a *probabilistic* algorithm on input σ, while $\Sigma^{\mathsf{asym}}.\mathsf{Enc}_{pk}(\sigma; \sigma')$ is a *deterministic* function of the two inputs σ and σ'.

   The final ciphertext is $C = C_1 \| C_2$.

3. When decrypting a ciphertext $C = C_1 \| C_2$ we first decrypt C_1 to get σ. Then we derive $\sigma' \leftarrow G(\sigma, C_2)$, and *re-encrypt* σ with Σ^{asym} using random coins σ'. If the result is not equal to the original C_1 we return $\perp$, else we derive K (from σ) and decrypt C_2 with Σ^{sym}.

The details of the FO-transform are given in Fig. 4.

a) Suppose the public-key encryption scheme Σ^{asym} has private/public-key space $\mathcal{SK} \times \mathcal{PK}$, message space $\mathcal{M}_1$ and ciphertext space $\mathcal{C}_1$; and that the symmetric encryption scheme Σ^{sym} has key space $\mathcal{K}$, message space $\mathcal{M}_2$ and ciphertext space $\mathcal{C}_2$. Then their

corresponding encryption algorithms have the following "type signatures":

$$\Sigma^{\mathsf{asym}}.\mathsf{Enc} : \mathcal{PK} \times \mathcal{M}_1 \to \mathcal{C}_1$$
$$\Sigma^{\mathsf{sym}}.\mathsf{Enc} : \mathcal{K} \times \mathcal{M}_2 \to \mathcal{C}_2$$

Similarly, their decryption algorithms have type signatures:

$$\Sigma^{\mathsf{asym}}.\mathsf{Dec} : \mathcal{SK} \times \mathcal{C}_1 \to \mathcal{M}_1$$
$$\Sigma^{\mathsf{sym}}.\mathsf{Dec} : \mathcal{K} \times \mathcal{C}_2 \to \mathcal{M}_2 \cup \{\perp\}.$$

What are the type signatures of FO.Enc and FO.Dec?

b) Show that the FO transform yields a correct encryption scheme. That is, show that $\mathsf{FO.Dec}(sk, \mathsf{FO.Enc}(pk, M)) = M$

c) Suppose your are using Textbook ElGamal as the public-key encryption scheme Σ^{asym} in the FO-transform. What happens if you carry out your attack from Problem 3 now?

d) It is possible to prove that the FO-transform gives an IND-CCA secure public-key encryption scheme provided that the public-key encryption scheme Σ^{asym} is IND-CPA secure[1], the symmetric encryption scheme Σ^{sym} is (one-time) IND-CCA secure, and the hash functions are modeled as *random oracles*[2]. Providing a formal proof of this fact is not so easy, however. Instead, try to give some high-level arguments for why an IND-CCA attacker against an FO-transformed public-key encryption scheme is unlikely to succeed.

References

[Ros] Mike Rosulek. *The Joy of Cryptography*. https://joyofcryptography.com.

[1] Plus an additional assumption on the distribution of the ciphertexts.

[2] A random oracle is simply a keyless *publicly accessibly* function that on input X responds with a random output Y. It returns the same value Y if queried on X again. However, the *internals* of the random oracle are completely hidden, i.e., the only way to learn an output value is by querying it on some input value, hence the name *oracle*. Modeling a hash function as a random oracle is a *very* strong assumption. Essentially, by invoking the random oracle model we are assuming that any attacker against the full construction (e.g. the FO-transform), will not try to exploit the internal structure of the hash functions.