

Introduction to Cryptography

TEK 4500 (Fall 2024)

Problem Set 9

Problem 1.

Read Chapter 10.3 and Chapter 11 in [BR] and Chapter 7 in [PP].

Problem 2.

Implement Textbook RSA in a programming language of your choice. Verify that your implementation achieves correctness: first encrypting with the public key and then decrypting the ciphertext with the private key should give back the original message.

Hint: Use Sage! This is basically Python, but with a lot of additional enhancements to deal with the algebraic structures used in cryptography. Some useful functions:

- `next_prime(n)` – return the first prime number larger than the integer n .
- `Integers(n)` – create the structure $\mathbf{Z}_n$. To create the elements 5 and 7 in $\mathbf{Z}_9$ write
 - 1: `Zn = Integers(n)`
 - 2: `a = Zn(5)`
 - 3: `b = Zn(7)`

If you then do

- 1: `a + b`
- 2: `a * b`

the result will be 3 and 8, respectively, which is the expected result in $\mathbf{Z}_9$. Note that you didn't explicitly have to do the $(\text{mod } 9)$ operation.

- One difference from Python: in Sage the `^` operation means exponentiation and not XOR as in Python.

Problem 3.

As noted in class, Textbook RSA should *not* be thought of as an encryption scheme in and of itself. The reason is that Textbook RSA is deterministic and thus has no chance of achieving

$\text{Exp}_{\Sigma}^{\text{ind-cca}}(\mathcal{A})$	$\mathcal{E}(M)$:	$\mathcal{D}(C)$:
1: $b \leftarrow \{0, 1\}$	1: $R \xleftarrow{\$} \{0, 1\}^{ M }$	1: if $C \in \text{Ciphertexts}$: // Cheating!
2: $\text{Ciphertexts} \leftarrow []$	2: $C_0 \leftarrow \Sigma.\text{Enc}(pk, R)$	2: return $\perp$
3: $(sk, pk) \xleftarrow{\$} \Sigma.\text{KeyGen}$	3: $C_1 \leftarrow \Sigma.\text{Enc}(pk, M)$	3: return $\Sigma.\text{Dec}(sk, C)$
4: $b' \leftarrow \mathcal{A}^{\mathcal{E}(\cdot), \mathcal{D}(\cdot)}(pk)$	4: $\text{Ciphertexts.add}(C_b)$	
5: return $b' \stackrel{?}{=} b$	5: return C_b	

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

Figure 1: IND-CCA security experiment for a public-key encryption scheme Σ .

IND-CPA security. Instead, Textbook RSA should be thought of as a more basic *primitive*, from which we can *build* an encryption scheme. One way of doing this is by padding the message with random bits before encrypting with Textbook RSA.

Consider the following padded version of RSA: for a modulus n of k bits, the message is a bit string of length $\ell < k$ for some *fixed* ℓ . When encrypting, the message $M \in \{0, 1\}^{\ell}$ is first padded with $k - \ell - 1$ ¹ random bits $P \in \{0, 1\}^{k-\ell-1}$. The concatenation $X = P \| M$ is then treated as an integer in the natural way and encrypted with Textbook RSA. On decryption, Textbook RSA decryption is applied and the first $k - \ell - 1$ bits are removed. The remaining bits are returned as the decrypted message.

For very small ℓ relative to k (e.g. $\ell \approx 10$ and $k = 2048$) it is possible to show that Padded RSA is IND-CPA secure under the RSA-assumption.

Exercise: Unfortunately, Padded RSA is *not* IND-CCA secure (ref Fig. 1). Show this.

Hint: Exploit the fact that RSA has the following property: if $C = M^e \pmod{n}$, then $S^e \cdot C = (S \cdot M)^e \pmod{n}$.

Problem 4.

Suppose you are given $n = p \cdot q$ and $\phi(n) = (p - 1)(q - 1) = n - p - q + 1$, where p and q are two distinct prime numbers.

- Find an expression for p (or q) in terms of n and $\phi(n)$.
- Suppose you are given $n = 1517$ and $\phi(n) = 1440$. Find p and q .
- Suppose you are given

$n = 0x58cfda78810ec57ec74cf45415cbd9ee386e775550e4a3654b62db2a9ca32f9ed6a9d0e6d8c85e7f0ba5cf4375fd68157b56329d1b2675$

and

$\phi(n) = 0x58cfda78810ec57ec74cf45415cbd9ee386e775550e4a3654b62db1582d94f712123656dc2ec8fba147f302523b7d045f9016c257bd76c$

Find p and q .

¹The “ -1 ” is just to ensure that the padded message is smaller than the modulus n

Problem 5.

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 $\text{KEM} = (\text{KeyGen}, \text{Encaps}, \text{Decaps})$, 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.
- $\text{Encaps} : \mathcal{EK} \rightarrow \mathcal{K} \times \mathcal{C}$ is a probabilistic *encapsulation* algorithm that, on input a public encapsulation key $ek \in \mathcal{EK}$, outputs a *symmetric* key $K \in \mathcal{K}$ and a ciphertext $C \in \mathcal{C}$. That is, $(K, C) \leftarrow \text{Encaps}(ek)$. Notice that Encaps does not take a message as input!
- $\text{Decaps} : \mathcal{DK} \times \mathcal{C} \rightarrow \mathcal{K}$ is a deterministic *decapsulation* algorithm that, on input a private decapsulation key $dk \in \mathcal{DK}$ and a ciphertext $C \in \mathcal{C}$, outputs a symmetric key $K \in \mathcal{K}$. That is, $K \leftarrow \text{Decaps}(dk, C)$.

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 .

- Formulate a *correctness* requirement for a KEM.
- Show how you can create a KEM scheme $\text{KEM} = (\text{KeyGen}, \text{Encaps}, \text{Decaps})$ from a public-key *encryption* scheme $\Sigma = (\text{KeyGen}, \text{Enc}, \text{Dec})$, where
 - $\Sigma.\text{Enc} : \mathcal{EK} \times \{0, 1\}^n \rightarrow \mathcal{C}$
 - $\Sigma.\text{Dec} : \mathcal{DK} \times \mathcal{C} \rightarrow \{0, 1\}^n$

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?

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

Hint: Make use of a *symmetric* encryption scheme $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$ 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 $b \xleftarrow{\$} \{1, \dots, |G|\}$, and defining $dk = b$ and $ek = g^b$, where g is the generator for the cyclic group 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.

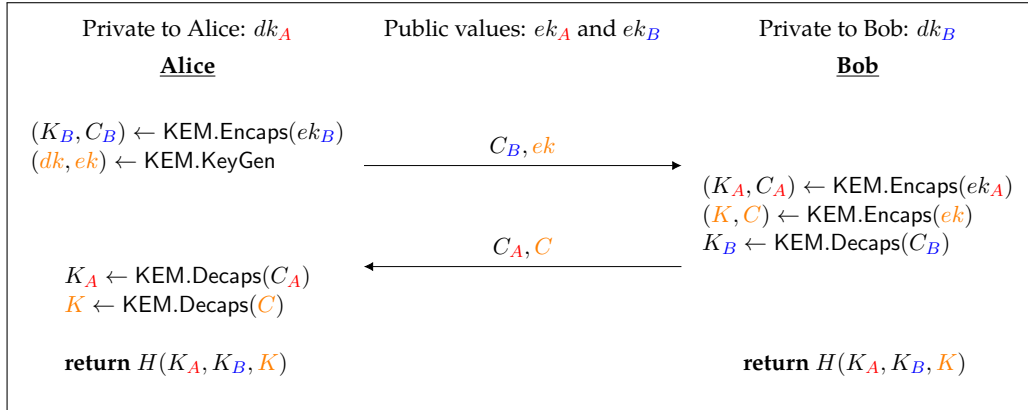


Figure 2: Authenticated key exchange from a KEM.

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 decapsulation 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 .

- 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 ?

Problem 6.

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. 3).

ElGamal.KeyGen:	ElGamal.Enc($pk = X, M$):
1: $x \xleftarrow{\$} \{0, 1, \dots, G - 1\}$	1: $y \xleftarrow{\$} \{0, 1, \dots, G - 1\}$
2: $X \leftarrow g^x$	2: $Y \leftarrow g^y$
3: return ($sk = x, pk = X$)	3: $Z \leftarrow X^y$
	4: $C \leftarrow Z \circ M$
	5: return (Y, C)

Figure 3: The Textbook ElGamal encryption scheme. It is parameterized by a cyclic group $G = \langle g \rangle$.

- 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. 1).

Problem 7.

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 \(FO\) transformation](#). The FO-transform consists of essentially three steps:

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

FO.KeyGen:	FO.Enc(pk, M):	FO.Dec(sk, C):
1: $(sk, pk) \xleftarrow{\$} \Sigma^{\text{asym}}.\text{KeyGen}$	1: $\sigma \xleftarrow{\$} \{0, 1\}^k$	1: Parse C as (C_1, C_2)
2: return (sk, pk)	2: $K \leftarrow H(\sigma)$	2: $\sigma \leftarrow \Sigma^{\text{asym}}.\text{Dec}(sk, C_1)$
	3: $C_2 \leftarrow \Sigma^{\text{sym}}.\text{Enc}(K, M)$	3: $K \leftarrow H(\sigma)$
	4: $\sigma' \leftarrow G(\sigma, C_2)$	4: $\sigma' \leftarrow G(\sigma, C_2)$
	5: $C_1 \leftarrow \Sigma^{\text{asym}}.\text{Enc}(pk, \sigma; \sigma')$	5: $M \leftarrow \Sigma^{\text{sym}}.\text{Dec}(K, C_2)$
	6: return C_1, C_2	6: $C'_1 \leftarrow \Sigma^{\text{asym}}.\text{Enc}(pk, \sigma; \sigma')$
		7: if $C'_1 = C_1$:
		8: return M
		9: else
		10: return $\perp$

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 .

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 $\Sigma^{\text{asym}}.\text{Enc}$. To make this explicit we use the notation $C_2 \leftarrow \Sigma^{\text{asym}}.\text{Enc}_{pk}(\sigma; \sigma')$, as opposed to the usual notation $C_2 \leftarrow \Sigma^{\text{asym}}.\text{Enc}_{pk}(\sigma)$ where the internal randomness is “hidden”. Thus, $\Sigma^{\text{asym}}.\text{Enc}_{pk}(\sigma)$ is a *probabilistic* algorithm on input σ , while $\Sigma^{\text{asym}}.\text{Enc}_{pk}(\sigma; \sigma')$ 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”:

$$\begin{aligned}\Sigma^{\text{asym}}.\text{Enc} : \mathcal{PK} \times \mathcal{M}_1 &\rightarrow \mathcal{C}_1 \\ \Sigma^{\text{sym}}.\text{Enc} : \mathcal{K} \times \mathcal{M}_2 &\rightarrow \mathcal{C}_2\end{aligned}$$

Similarly, their decryption algorithms have type signatures:

$$\begin{aligned}\Sigma^{\text{asym}}.\text{Dec} : \mathcal{SK} \times \mathcal{C}_1 &\rightarrow \mathcal{M}_1 \\ \Sigma^{\text{sym}}.\text{Dec} : \mathcal{K} \times \mathcal{C}_2 &\rightarrow \mathcal{M}_2 \cup \{\perp\}.\end{aligned}$$

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 $\text{FO.Dec}(sk, \text{FO.Enc}(pk, M)) = M$
- c) Suppose you 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 6 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 secure², the symmetric encryption scheme Σ^{sym} is (one-time) IND-CCA secure, and the hash functions are modeled as *random oracles*³. 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

- [BR] Mihir Bellare and Phillip Rogaway. *Introduction to Modern Cryptography*. <https://web.cs.ucdavis.edu/~rogaway/classes/227/spring05/book/main.pdf>.
- [PP] Christof Paar and Jan Pelzl. *Understanding Cryptography - A Textbook for Students and Practitioners*. Springer, 2010.

²Plus an additional assumption on the distribution of the ciphertexts.

³A 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.