

# Introduction to Cryptography

TEK 4500 (Fall 2022)

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  $\wedge$  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

```

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

 $\mathcal{E}(M)$ :
1:  $R \xleftarrow{\$} \{0, 1\}^{|M|}$ 
2:  $C_0 \leftarrow \Sigma.\text{Enc}(pk, R)$ 
3:  $C_1 \leftarrow \Sigma.\text{Enc}(pk, M)$ 
4: Ciphertexts.add( $C_b$ )
5: return  $C_b$ 

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

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

```

**Figure 1:** IND-CCA security experiment for a public-key encryption scheme  $\Sigma$ .

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 space is bit strings of  $\ell < k$  bits for some *fixed*  $\ell$ . When encrypting, the message  $M \in \{0, 1\}^{\ell}$  is first padded with  $k - \ell - 1$  random bits  $P \in \{0, 1\}^{k-\ell-1}$ .<sup>1</sup> 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  $\ell$  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  $S^e \cdot C = (S \cdot M)^e \pmod{n}$ .

<sup>1</sup>The “ $-1$ ” is just to ensure that the padded message is smaller than the modulus  $n$

#### 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.

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

$$n = 0x58cfda78810ec57ec74cf45415cbd9ee386e775550e4a3654b62db2a9ca32f9ed6a9d0e6d8c85e7f0ba5cf4375fd68157b56329d1b2675$$

and

$$\phi(n) = 0x58cfda78810ec57ec74cf45415cbd9ee386e775550e4a3654b62db1582d94f712123656dc2ec8fba147f302523b7d045f9016c257bd76c$$

Find  $p$  and  $q$ .

#### Problem 5.

In practice, whenever RSA encryption is used (in some properly padded form; see Problem 3), it is only used to encrypt a short symmetric key. This key is then used in some symmetric encryption scheme to encrypt the actual data. Thus, RSA encryption is in reality mostly used as a *key transport mechanism* of symmetric keys. We've already seen another way of establishing a shared key between two parties: the Diffie-Hellman key exchange protocol. Thus, we have two natural ways for Alice and Bob to establish a shared secret between them:

- Diffie-Hellman: Alice and Bob run the Diffie-Hellman protocol.
- RSA: Alice picks a random symmetric key and then encrypts it with Bob's RSA public key. The ciphertext of the key is sent to Bob which decrypts it to obtain the key.

- a) Compare these two methods for establishing a shared secret. Focus both on security and efficiency.

**Hint:** Look up the story of the email service provider Lavabit and why it was shut down in August 2013.

**Hint:** A keyword is [forward secrecy](#).

- b) Explain how you would obtain forward secrecy when using RSA for key exchange.

#### 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  $\Sigma$ . Instead, they encrypt the message directly using the derived key  $Z$  in a "one-time-pad" like fashion (see Fig. 2).

<u>ElGamal.KeyGen:</u>	<u>ElGamal.Enc(pk = X, M):</u>
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: <b>return</b> ( $sk = x, pk = X$ )	3: $Z \leftarrow X^y$
	4: $C \leftarrow Z \circ M$
	5: <b>return</b> ( $Y, C$ )

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

- What is the message space for Textbook ElGamal?
- Describe the decryption algorithm for Textbook ElGamal.
- 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  $\Sigma^{\text{asym}}$  into an IND-CCA secure one is to apply something called the [Fujisaki-Okamoto \(FO\) transformation](#). The FO-transform consists of essentially three steps:

- Generate a random bitstring  $\sigma$ . From  $\sigma$  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  $\Sigma^{\text{sym}}$ , yielding a ciphertext  $C_2$ .
- Encrypt  $\sigma$  with the IND-CPA secure public-key encryption scheme  $\Sigma^{\text{asym}}$ , 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  $\sigma'$  are derived from  $\sigma$  and  $C_2$  using another hash function  $G$ , i.e.  $\sigma' \leftarrow G(\sigma, C_2)$ .

In particular, when encrypting  $\sigma$  we use  $\sigma'$  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  $\sigma$ , while  $\Sigma^{\text{asym}}.\text{Enc}_{pk}(\sigma; \sigma')$  is a *deterministic* function of the two inputs  $\sigma$  and  $\sigma'$ .

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

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

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: <b>return</b> $(sk, pk)$	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: <b>return</b> $C_1, C_2$	6: $C'_1 \leftarrow \Sigma^{\text{asym}}.\text{Enc}(pk, \sigma; \sigma')$
		7: <b>if</b> $C'_1 = C_1$ :
		8: <b>return</b> $M$
		9: <b>else</b>
		10: <b>return</b> $\perp$

**Figure 3:** The FO-transform. It is parameterized by a public-key encryption scheme  $\Sigma^{\text{asym}}$ , a symmetric encryption scheme  $\Sigma^{\text{sym}}$ , and two hash functions  $H, G$ .

The details of the FO-transform are given in Fig. 3.

- a) Suppose the public-key encryption scheme  $\Sigma^{\text{asym}}$  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  $\Sigma^{\text{sym}}$  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  $\Sigma^{\text{asym}}$  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  $\Sigma^{\text{asym}}$  is IND-CPA secure<sup>2</sup>, the symmetric encryption scheme  $\Sigma^{\text{sym}}$  is (one-time) IND-CCA secure, and

<sup>2</sup>Plus an additional assumption on the distribution of the ciphertexts.

the hash functions are modeled as *random oracles*<sup>3</sup>. 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.

---

<sup>3</sup>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.