Introduction to Cryptography

TEK 4500 (Fall 2021)
Problem Set 10

Problem 1.

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

Problem 2.

Textbooks often present a simpler variant of ElGamal than what we did in class. In particular, *Textbook* ElGamal differs from the version we looked at in the following ways.

a) The message space is simply the group G.

b) It does not use a hash function to derive the encryption key.

c) It does not use a general purpose symmetric encryption scheme to encrypt the message. Instead, it simply multiplies[1] the Diffie-Hellman secret Z with the message M directly.

The details of Textbook ElGamal are given in Fig. 1. Note that all the elements M, X, Y, Z, C, C' are elements in the cyclic group $(G, *) = \langle g \rangle$.

Suppose we use Textbook ElGamal with the group $(\mathbf{Z}^*_{154943}, \cdot)$ and using 5 as the generator.

a) Let $sk = 51237$. Calculate pk.

b) Let $M = 102400$. Encrypt this message (i.e. calculate (Y, C)) assuming we pick $y = 6789$ during encryption.

c) Verify that you are able to decrypt $C' = (Y, C)$ using sk.

Hint: Use Sage! (no installation required; simply run your code in a web browser). Sage is basically Python, but with a lot of additional enhancements to deal with the algebraic structures used in cryptography. Some useful functions:

[1]Here "multiply" means the $*$ operation in the group $(G, *)$.

ElGamal.KeyGen:

$x \xleftarrow{\$} \{0, 1, \ldots, |G| - 1\}$
$X \leftarrow g^x$
return $(sk = x, pk = X)$

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

$y \xleftarrow{\$} \{0, 1, \ldots |G| - 1\}$
$Y \leftarrow g^y$
$Z \leftarrow X^y$
$C \leftarrow Z * M$
return (Y, C)

ElGamal.Dec$(sk = x, C')$**:**

Parse C' as (Y, C)
$Z \leftarrow Y^x$
$M \leftarrow Z^{-1} * C$
return M

Figure 1: The Textbook ElGamal encryption scheme. It is parameterized by a cyclic group $G = \langle g \rangle$. Note that the message space is G, i.e., the messages are group elements.

- `is_prime(n)` – check if n is a prime number.
- `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

  ```
  Zn = Integers(n)
  a = Zn(5)
  b = Zn(7)
  ```

  If you then do

  ```
  a + b
  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 $\pmod 9$ operation.
- `FiniteField(p)` – create the finite field $\mathbf{F}_p$ over the prime p. Can be used the same way as for `Integers(n)`.
- One difference from Python: in Sage the ^ operation means exponentiation and not XOR as in Python.

Suppose we now use Textbook ElGamal with the elliptic curve group $(E(\mathbf{F}_{154943}), +)$, where E is the elliptic curve

$$E : y^2 = x^3 + 3x + 6$$

defined over the finite field $\mathbf{F}_{154943}$. In the following exercises it is highly recommended to use Sage. In particular, to define the group $(E(\mathbf{F}_{154943}), +)$ above, write the following in Sage.

```
p = 154943
F = FiniteField(p)
a = 3
b = 6
E = EllipticCurve(F, [a,b])
print(E)
print(E.order())         # count the number of points on E
P = E.random_point()     # pick a random point on the curve
Q = E.random_point()
print(P)
print(Q)
print(P + Q)             # perform elliptic curve addition
print(100 * P)           # add P to itself 100 times
R = E(124599,36054)      # define a point with explicit (x,y) coordinates
```

d) Verify that the point

```
P = E(138357,2620)
```

is a generator for the group $(E(\mathbf{F}_{154943}), +)$.

e) Let $sk = 51237$. Calculate pk.

f) Let $M = (64356, 90882)$. Encrypt this message (i.e. calculate (Y, C)) assuming we pick $y = 6789$ during encryption.

g) Verify that you are able to decrypt $C' = (Y, C)$ using sk.

Problem 3.

Show that Textbook ElGamal (Fig. 1) does not achieve IND-CCA security (ref Fig. 2).

Problem 4.

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!

Problem 5.

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 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.

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

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

$\underline{\mathcal{E}(M)}$:

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: $\text{Ciphertexts.add}(C_b)$
5: **return** C_b // real ciphertext or encryption of random string

$\underline{\mathcal{D}(C)}$:

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

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

Figure 2: IND-CCA security experiment.

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* ℓ. When encrypting, the message $M \in \{0,1\}^\ell$ is first padded with $k - \ell - 1$ random bits $R \in \{0,1\}^{k-\ell-1}$. The concatenation $X = R \| M$ 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. However, Padded RSA is *not* IND-CCA secure (ref Fig. 2). **Exercise:** 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 6.

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) Given

$n =$ `0x58cfda78810ec57ec74cf45415cbd9ee386e775550e4a3654b62db2a9ca32f9ed6a9d0e6d8c85e7f0ba5cf4375fd68157b56329d1b2675`

and

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

Find p and q.

Problem 7.

In practice, whenever RSA encryption is used (in some properly padded form; see Problem 5), 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 8.

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:

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^{asym}.

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

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

$\underline{\textsf{FO.KeyGen}}$:

$(sk, pk) \xleftarrow{\$} \Sigma^{\textsf{asym}}.\textsf{KeyGen}$
return (sk, pk)

$\underline{\textsf{FO.Enc}(pk, M)}$:

$\sigma \xleftarrow{\$} \{0,1\}^k$
$K \leftarrow H(\sigma)$
$C_2 \leftarrow \Sigma^{\textsf{sym}}.\textsf{Enc}(K, M)$
$\sigma' \leftarrow G(\sigma, C_2)$
$C_1 \leftarrow \Sigma^{\textsf{asym}}.\textsf{Enc}(pk, \sigma; \sigma')$
return C_1, C_2

$\underline{\textsf{FO.Dec}(sk, C)}$:

Parse C as (C_1, C_2)
$\sigma \leftarrow \Sigma^{\textsf{asym}}.\textsf{Dec}(sk, C_1)$
$K \leftarrow H(\sigma)$
$\sigma' \leftarrow G(\sigma, C_2)$
$M \leftarrow \Sigma^{\textsf{sym}}.\textsf{Dec}(K, C_2)$
$C_1' \leftarrow \Sigma^{\textsf{asym}}.\textsf{Enc}(pk, \sigma; \sigma')$
if $C_1' = C_1$**:**
    return M
else
    return $\perp$

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

scheme $\Sigma^{\textsf{sym}}$ 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":

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

Similarly, their decryption algorithms have type signatures:

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

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 $\textsf{FO.Dec}(sk, \textsf{FO.Enc}(pk, M)) = M$

c) Suppose your are using Textbook ElGamal as the public-key encryption scheme $\Sigma^{\textsf{asym}}$ 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 $\Sigma^{\textsf{asym}}$ is IND-CPA secure[2], the symmetric encryption scheme $\Sigma^{\textsf{sym}}$ 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.

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

[3]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

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.

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.