Lecture 7 - Public key encryption

Gidon Rosalki

2025-12-17

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_intro_to_crypto`

1 Public key encryption

1.1 Definitions

Public key encryption involves the following syntax: There are three algorithms $\Pi = (KeyGen, Enc, Dec)$:

- The key generation algorithm $KeyGen\left(1^n\right)$ outputs a secret key sk, and a public key pk
- Encryption algorithm Enc takes a public key pk and a plaintext m, and outputs a ciphertext c
- Decryption algorithm Dec takes a secret key sk and a ciphertext c, and outputs a plaintext m

It is correct if for every $m \in \mathcal{M}$

$$\Pr\left[Dec_{sk}\left(Enc_{pk}\left(m\right)\right) = m\right] = 1$$

Definition 1.1 (IND-CPA)**.** Π *has indistinguishable encryptions under a chosen-plaintext attack if for every PPT adversary* $\mathcal{A}$ *there exists a negligible function* $v\left(\cdot\right)$ *such that*

$$\Pr\left[IND_{\Pi,\mathcal{A}}^{CPA}\left(n\right) = 1\right] \leq \frac{1}{2} + v\left(n\right)$$

So here the game is subtly different. Recall that the game is that $\mathcal{A}$ receives the encryption algorithm, produces two plaintexts, receive the encryption of one of them, and has to guess which one. Now, instead of the encryption algorithm, it receives the public key. There is no need to provide an encryption oracle, since $\mathcal{A}$ can encrypt by itself using the public key. However, one **must** use a randomised encryption, to avoid $\mathcal{A}$ just encrypting m_0, m_1 by itself. In comparison, for CCA $\mathcal{A}$ can also access a decryption oracle (aside from for c^*).

1.2 Encrypting long messages

So we need to encrypt long messages. This can be done by encrypting them in blocks:

$$Enc'_{pk}\left(m^{(1)}\ldots m^{p(n)}\right) = \left(Enc_{pk}\left(m^{(1)}\right), \ldots, Enc_{pk}\left(m^{p(n)}\right)\right)$$

Theorem 1. *If* $\Pi = (KeyGen, Enc, Dec)$ *is CPA secure, then for any polynomial* $p\left(n\right)$ *the scheme* $\Pi' = (KeyGen, Enc', Dec')$ *is CPA secure*

Proof Idea. Given an adversary $\mathcal{A}'$ for Π', construct an adversary $\mathcal{A}$ for Π. $\mathcal{A}$ gets one challenge ciphertext, and generates the others on its own.

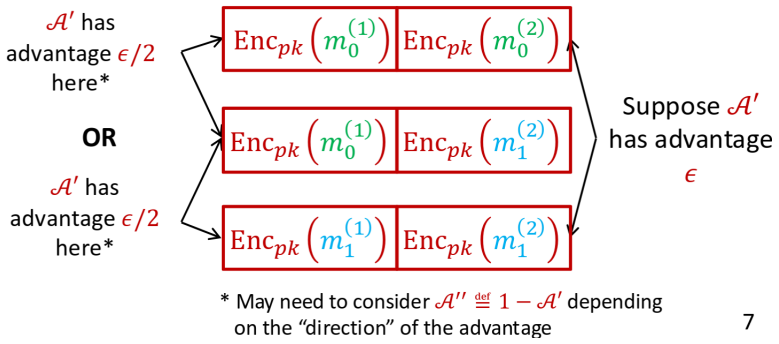

Figure 1: Hybrid argument

So $\mathcal{A}$ can generate the messages $\left(m_0^{(1)}, m_0^{(2)}\right)$, and $\left(m_1^{(1)}, m_1^{(2)}\right)$, and mix the two halves. It can then (for example) provide $\left(m_0^{(1)}, m_0^{(2)}\right)$ and $\left(m_0^{(1)}, m_1^{(2)}\right)$, with an advantage of at least $\frac{\varepsilon}{2}$. Since it can differentiate between the halves, it is then able to differentiate between the messages. □

Now, public key schemes are somewhat inefficient, and slow to compute, but symmetric (private key) schemes are much faster. The solution is called **Hybrid encryption**, where one generates a *session key* k, encrypts it (since it is short) with a public key encryption scheme, and then encrypt m with a private key scheme, using k as the key.

2 Hybrid encryption

Definition 2.1 (Hybrid encryption)**.** *Let $\Pi = (KeyGen, Enc, Dec)$ be a public key encryption scheme, and let (E, D) be a symmetric key encryption scheme. We can now define a public key encryption scheme $\Pi' = (Gen', Enc', Dec')$, where*

- $Gen' = Gen$
- $Enc'_{pk}(m)$ *samples* $k \leftarrow \{0,1\}^n$*, computes* $c_1 \leftarrow Enc_{pk}(k)$*, and* $c_2 \leftarrow E_k(m)$*. It outputs* $c - (c_1, c_2)$
- $Dec'_{sk}(c_1, c_2)$*: Compute* $k \leftarrow Dec_{sk}(c_1)$*, and output* $D_k(c_2)$

Theorem 2. *If Π is CPA-secure, and (E, D) is IND-secure, then Π' is CPA secure*

Proof . Let there be $\mathcal{A}'$ for Π'. It receives pk, outputs m_0, m_1, receives $Enc_{pk}(k), E_k(m_b)$, and outputs b'. From this we may construct an adversary $\mathcal{A}$ for Π, or an adversary $\mathcal{B}$ for (E, D). Taking the two messages that $\mathcal{A}'$ outputs, m'_0, m'_1, then we may consider the following:

1. $(Enc_{pk}(k), E_k(m'_0))$
2. $(Enc_{pk}(0^n), E_k(m'_0))$
3. $(Enc_{pk}(0^n), E_k(m'_1))$
4. $(Enc_{pk}(k), E_k(m'_1))$

From 1 - 4 $\mathcal{A}'$ let us say that $\mathcal{A}'$ has the advantage ε. As a result, there are 2 stages where it has an advantage of at least $\frac{\varepsilon}{3}$. We can now build $\mathcal{A}$, that simulates $\mathcal{A}'$. Upon receiving m'_0, m'_1, we will output

$$m_0 = k$$
$$m_1 = 0^n$$

and receive in return the public key encryption of one of them $c^* = Enc_{pk}(m_b)$. We can then then return $(c^*, E_k(m'_0))$ to $\mathcal{A}'$, which will output the correct b' with its advantage, since it can differentiate between $Enc_{pk}(k)$ and $Enc_{pk}(0^n)$. Conversely, if we set

$$m_0 = m'_0 m_1 = m'_1 \tag{1}$$

Then we can give $\mathcal{A}'$ the input $(Enc_{pk}(0^n), c^*)$, where $c^* = E_k(m_b)$, and since it has the advantage for $E_k(m'_0)$ and $E_k(m'_1)$, we will once again get back the correct b'.
Finally, we can also have $\mathcal{A}$ output

$$m_0 = 0^n$$
$$m_1 = k$$

Receiving back $c^* = Enc_{pk}(m_b)$, give $(c^*, E_k(m'_1))$ to $\mathcal{A}'$, and thanks to the advantage that $\mathcal{A}'s$ has between $Enc_{pk}(0^n)$ and $Enc_{pk}(k)$, it will return the correct b'. □

2.1 El-Gamal Encryption

Behold a real public key encryption scheme. It is based on Diffie-Hellman key agreement, and relies on the DDH assumption. Recall the DDH assumption: Let $\mathcal{G}$ be a PPT algorithm that on input 1^n, outputs $(\mathbb{G}, q, g)$, where $\mathbb{G}$ is a cyclic group of order q, that is generated by g, and q is an $n$bit prime.

Definition 2.2 (The Decisional Diffie Hellman (DDH) Assumption)**.** *For every PPT algoirithm $\mathcal{A}$ there exists a negligible function $v(\cdot)$ such that*

$$|\Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^{xz}) = 1] - \Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^z) = 1]| \leq v(n)$$

Where $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n)$, and $x, y, z \leftarrow \mathbb{Z}_q$

So: Let $\mathcal{G}$ be a PPT algorithm that on input 1^n outputs $(\mathbb{G}, q, g)$, where $\mathbb{G}$ is a cyclic group of order q, that is generated by g. We will define a public key encryption scheme $\Pi = (KeyGen, Enc, Dec)$ as

- $Gen'(1^n)$: Sample $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n)$, and $x \leftarrow \mathbb{Z}_q$. Let $h = g^x$, and output $pk = (\mathbb{G}, q, g, h)$, and $sk = x$
- $Enc_{pk}(m)$: Sample $y \leftarrow \mathbb{Z}_q$, and output $c = (g^y, h^y \cdot m)$

- $Dec_{sk}(c_1, c_2)$: Output $m = \frac{c_2}{c_1^x}$

Note that there are methods for encoding binary strings as group elements, and that for simplicity, we assume that the plaintext set is $\mathbb{G}$.

$$Dec_{sk}(Enc_{pk}(m)) = Dec_{sk}(g^y, h^y \cdot m) = \frac{h^y \cdot m}{(g^y)\, tox} = \frac{(g^x)^y \cdot m}{(g^y)^x} = m$$

Theorem 3 (Security)**.** *Under the DDH assumption, the scheme* Π *is secure*

Proof . Hey look! Another reduction. We will assume that there exists $\mathcal{A}$ that breaks Π, and so we can build $\mathcal{D}$ that breaks DDH. So, $\mathcal{D}$ receives (g^x, g^y, g^z), and needs to return if $z = xy$, or if z is random.

Let us construct $\mathcal{D}$, that receives $(\mathbb{G}, q, g, g_1, g_2, g_3)$. It will give $\mathcal{A}$ $pk = (\mathbb{G}, q, g, g_1)$. $\mathcal{A}$ will return m_0, m_1, and then $\mathcal{D}$ will return to $\mathcal{A}$ the ciphertext $c^* = (g_2, g_3 \cdot m_b)$.

Case I:

$$(g, g_1, g_2, g_3) = (g, g^x, g^y, g^{xy})$$

Here, $\mathcal{A}$'s view is identical to the CPA experiment, and therefore

$$\Pr[\mathcal{D}(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1] = \Pr\left[IND^{CPA}_{\Pi,\mathcal{A}}(n) = 1\right]$$

Case II: Let z be a random number, of appropriate size:

$$(g, g_1, g_2, g_3) = (g, g^x, g^y, g^z)$$

The view of $\mathcal{A}$ is independent of the bit b, and so

$$\Pr[\mathcal{D}(\mathbb{G}, q, g, g^x, g^y, g^z) = 1] = \frac{1}{2}$$

So, given $(\mathbb{G}, q, g, g_1, g_2, g_3)$, our algorithm $\mathcal{D}$ will generate $pk = (\mathbb{G}, q, g, g_1)$, and upon $\mathcal{A}$ response of m_0, m_1, it will return $c^* = (g_2, g_3 \cdot m_b)$. It will take $\mathcal{A}$'s output b', and output 1 if $b' = b$, and 0 otherwise. By the DDH assumption:

$$\begin{aligned} v(n) &\geq |\Pr[\mathcal{D}(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1] - \Pr[\mathcal{D}(\mathbb{G}, q, g, g^x, g^y, g^z) = 1]| \\ &= \left|\Pr\left[IND^{CPA}_{\Pi,\mathcal{A}}(n) = 1\right] - \frac{1}{2}\right| \end{aligned}$$

□

2.2 RSA encryption

2.2.1 The RSA assumption

Let GenRSA be a PPT algorithm that on input 1^n outputs (N, e, d), where p, q are n bit primes, $N = pq$, $gcd(e, \phi(N)) = 1$, and $d = e^{-1} \mod \phi(N)$. Here $\phi(N)$ is the order of our set $\mathbb{Z}^*_{N=pq}$, such that $\phi(N) = (p-1)(q-1)$.

Definition 2.3 (RSA Assumption)**.** *For every PPT* $\mathcal{A}$ *there exists a negligible function* $v(\cdot)$ *such that*

$$\Pr[\mathcal{A}(N, e, x^e \mod N) = x] \leq v(n)$$

Where $(N, e, d) \leftarrow GenRSA(1^n)$ *and* $x \leftarrow \mathbb{Z}^*_N$.

In short, given $x^e \mod N$, it is very hard to compute x. In other words, $f_e(x) = x^e \mod N$ is a one way permutation family. f_d is the inverse of f_e, since $ed = 1 \mod \phi(N)$

2.2.2 Textbook RSA encryption

Let GenRSA be a PPT algorithm that on input 1^n outputs (N, e, d), where p, q are n bit primes, $N = pq$, $gcd(e, \phi(N)) = 1$, and $d = e^{-1} \mod \phi(N)$.

From here we create the public key $pk = (N, e)$, and the private key $sk = d$. Thus:

$$Enc_{pk}(m) = m^e \mod N \tag{2}$$

$$Dec_{sk}(c) = c^d \mod N \tag{3}$$

This is... Not a great system to be honest. It was first suggested in 1977, but the security definitions, like CPA-security were created in 1982. We may firstly note that Enc is deterministic, which is immediately **bad**. We will note that it is also not CPA-secure, many attacks are known. For example, if $m^e < N$, then $c = [m^e \mod N] = m^e$, and so $c^{\frac{1}{e}} = m$.

We may take this moment to state **NEVER USE TEXTBOOK RSA!!!**
To emphasise this point I have used atrocious grammar. I hope this helps you remember this.
(There is in fact an argument, which I will link `here`, that RSA should never be used. It is not a part of the course, and I have added it purely for your own interest.)

2.2.3 PKCS

Version 1.5 was standard issued by RSA labs in 1993. The idea is random padding:

$$pk = (N, e) \tag{4}$$
$$sk = d \tag{5}$$

So, $Enc_{pk}(m) = (r\|m)^e \mod N$, for a freshly chosen random r. This has the drawbacks that no proof of CPA security exists (aside from if m is very short). Chosen plaintext attacks are known if r is too short, and chosen ciphertext attacks are also known. In short, we do not know if it is secure, nor do we even have a neat little assumption (like DDH) that *if* it holds, we know it to be secure.

Next is version 2.0, which uses a more structured padding: Optimal Asymmetric Encryption Padding (OAEP). OAEP introduces redundancy, so that not every $c \in \mathbb{Z}_n^*$ is a valid ciphertext. This means that $Dec_{sk}(\cdot)$ must check for proper formatting upon decryption, and reject if it does not exist. This can be proved to be CCA-secure under the RSA assumption, if G and H are modelled as "random" hash functions. It is widely used in practice.

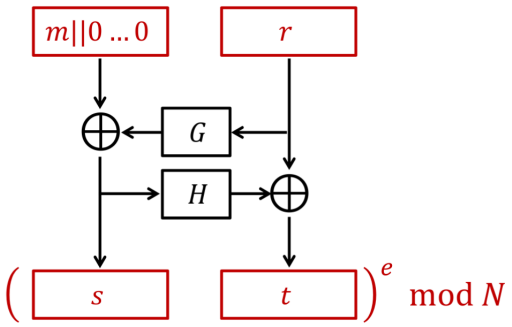

Figure 2:

Note that the RSA permutation family is **not** a CPA-secure PKE scheme. It is however a family of *trapdoor permutations*, which are one way permutations that may be efficiently inverted using a trapdoor. We will show a generic construction of a CPA secure scheme from any TDP (trapdoor permutation) family:

Definition 2.4 (Trapdoor permutation family)**.** *A tuple* $(Gen, Samp, f, f^{-1})$ *of PPT algorithms is a* ***trapdoor permutation family*** *if:*

- $Gen(1^n)$ *outputs pairs* (I, td) *defining a domain* $\mathcal{D}_I$
- $(Gen_1, Samp, f)$ *is a one way permutation family, where* Gen_1 *is obtained from* Gen *by outputting only* I
- f^{-1} *is deterministic, and for all* (I, td) *and* $x \in \mathcal{D}_I$ *it holds that* $f_{td}^{-1}(f_I(x)) = x$

For simplicity, we will typically write $x \leftarrow \mathcal{D}_I$ instead of $x \leftarrow \text{Samp}_I$, and (Gen, f, f^{-1}) instead of $(\text{Gen}, \text{Samp}, f, f^{-1})$. To summarise our cryptographic primitives once more:

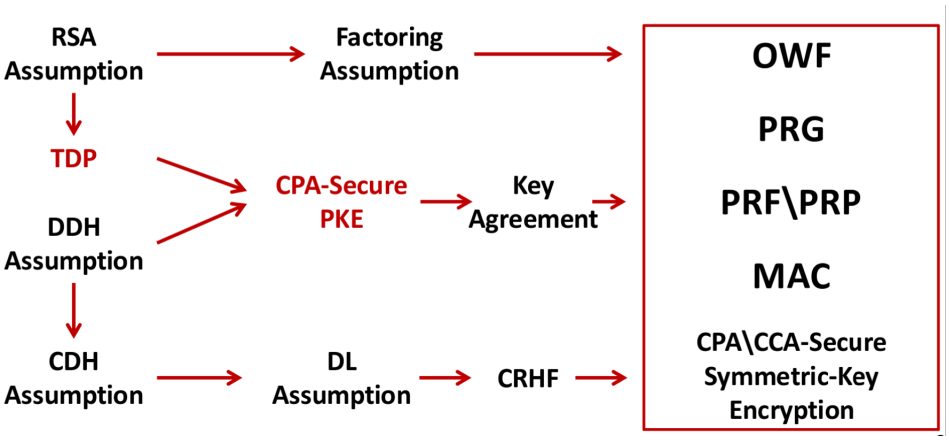

Figure 3: Cryptographic primitives

3 Constructions