

## Lecture 11: Semantically Secure Public-Key Encryption I

Instructor: Shafi Goldwasser

Scribes: Vahid Fazel-Rezai, Daniel Richman, Connor Sell

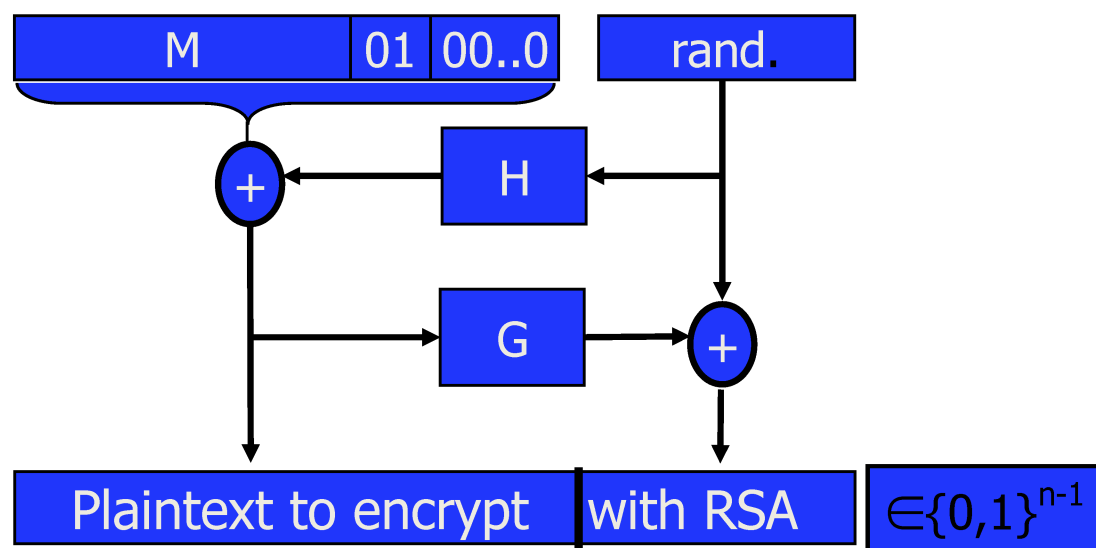
Today's lecture focused on implementing *public-key encryption*, especially RSA, and *bit-by-bit encryption*. We also discussed techniques for implementing *homomorphic encryption*, which enables some computation to be performed on encrypted data.

## 1 RSA preprocessing

RSA has many nice properties. For example, every bit of the input is a hard-core bit for the trapdoor function family. But last lecture we also observed some flaws in the plain vanilla RSA encryption scheme:

- Revealed information: for example, the Jacobi symbol of the ciphertext is equal to the Jacobi symbol of the message
- Determinism: RSA does not involve randomness, so a given message  $m$  always encrypts to the same ciphertext  $c$ . This makes RSA insecure against chosen-ciphertext attacks

How can we fix these flaws? One approach is to nondeterministically preprocess the message before encrypting it with RSA. One scheme is Optimal Asymmetric Encryption Padding (OAEP) [BR94]:



**Figure 1:** The OAEP block diagram

Theorem: OAEP is *semantically secure* when  $H, G$  are *random oracles*. Semantic security means that no partial information (other than the length of the message) is revealed by the ciphertext. The security holds under chosen-plaintext attacks and, when OAEP is used with RSA, under chosen-ciphertext attacks (CCA1) also.

## 2 Random Oracles in Theory and Practice

Real random oracles don't exist. But the idea is nice: a random oracle  $H$  produces outputs  $H(x)$  that are truly random but consistent. That is, the oracle always answers any query the same way. We model a random oracle as a function chosen truly randomly from the set of all possible functions [CGH04], where "possible functions" are defined as mapping  $\{0, 1\}^* \rightarrow \{0, 1\}^l$ , where  $l$  is the desired output length.

In practice, random oracles are often implemented using cryptographic hash functions such as SHA1. But systems whose cryptographic security proofs rely on the random oracle model are often not as secure when a hash function is used. Indeed, Canetti, Goldreich, and Halevi exhibited a crypto-system proven secure under the random oracle model but insecure with any particular implementation of the oracle [CGH04].

## 3 Implementation Attacks on RSA

OAEP is important because of vanilla RSA's insecurities. These include both theoretical faults and side-channel attacks on actual software implementations of RSA.

Several theoretical ways exist to model an adversary. Adversaries may be passive eavesdroppers on the communications channel; they may be able to encrypt messages of their choice (always true in public-key cryptography), in a chosen-plaintext attack; or they may even be able to decrypt ciphertexts of their choice, in a chosen-ciphertext or lunchtime attack.

In a lunchtime (CCA1) attack, the adversary, Mallory, can request encryptions and decryptions of arbitrary strings (while Alice, the challenger goes to lunch and leaves her computer unattended). Mallory then chooses two messages; Alice selects one at random and encrypts it, and Mallory must say which it is. Because vanilla RSA (without OAEP) is deterministic, Mallory can recognize messages she has already seen, and therefore her task is easy.

We can formalize the chosen-ciphertext attack as

$$\Pr \left[ \begin{array}{l} K \leftarrow \text{Gen}(1^n) \\ (M_0, M_1, \text{state}) \leftarrow \mathcal{A}^{\text{Enc}_K(\cdot), \text{Dec}_K(\cdot)}(1^n) \\ B \sim \mathcal{U}(\{0, 1\}) \\ C^* \leftarrow \text{Enc}(K, M_B) \\ B' \leftarrow \mathcal{A}^{\text{Enc}_K(\cdot)}(1^n, C^*, \text{state}) \end{array} : B = B' \right] < \frac{1}{2} + \text{negl}(n). \quad (1)$$

A number of attacks on RSA implementations also exist. These include:

- Timing attacks, where the number of CPU cycles required to compute  $c^d \pmod{n}$  reveals  $d$
- Power attacks, where CPU power consumption during computation of  $c^d \pmod{n}$  reveals  $d$
- Fault attacks, where an error at a critical moment reveals  $d$
- Cache attacks, where patterns of CPU cache access reveal  $d$

These are side-channel attacks, where information is revealed by a leaky implementation. Don't roll your own crypto!

## 4 Public key cryptography based on squaring

One other example of a public key cryptosystem, aside from RSA, is based on the squaring function. More precisely, given a large number  $N$  that is the product of two primes that are each 3 (mod 4), it is hard to find the square root of a given perfect square. (Perfect squares are also called quadratic residues.) However, it is easier to find such a root given the prime factors  $p$  and  $q$ . This allows us to create the following encryption scheme:

- Key generation - Generate two large primes  $p$  and  $q$  such that  $p \cong q \cong 3 \pmod{4}$ . Keep  $(p, q)$  as the secret key, but announce the public key  $N = pq$ .
- Encryption -  $\text{Enc}(N, m) = c = m^2 \pmod{N}$
- Decryption -  $\text{Dec}(N, c)$  gives the unique square root of  $c$  that is itself a perfect square modulo  $N$ .

If we know the prime factors  $p$  and  $q$ , we can easily find the roots of  $m^2$  modulo  $N$  by using the Chinese remainder theorem to decompose the problem into finding the roots of  $m^2$  modulo  $p$  and modulo  $q$ . We know how to find roots modulo primes in polynomial time (see Cipolla's algorithm), and can tell which roots are squares, so we can put this information back together to efficiently find the square root of  $m^2$  modulo  $N$  that is itself a perfect square.

However, if we don't know  $p$  and  $q$ , decrypting is as hard as factorizing  $N$ . In fact, we can find the prime factorization of  $N$  from the roots. As a result, this encryption scheme is especially vulnerable to lunchtime attacks, which we mentioned earlier. We will now provide an example of a successful lunchtime attack.

Suppose an attacker had access to the decryption oracle for a short time and found the roots of  $k^2$  and  $(jk)^2$  that were quadratic residues, for some  $j$  and  $k$  in  $\mathbb{Z}_N^*$ . Suppose  $j$  happened to be a quadratic residue modulo  $p$  but not modulo  $q$ . The roots of  $k^2$  modulo  $N$  correspond to pairs of roots modulo  $p$  and  $q$ , namely  $(k_p, k_q)$ ,  $(k_p, -k_q)$ ,  $(-k_p, k_q)$ , and  $(-k_p, -k_q)$ . Without loss of generality, say  $k_p$  and  $k_q$  (as opposed to  $-k_p$  and  $-k_q$ ) are quadratic residues modulo  $p$  and  $q$ , respectively. Then since  $j$  is a quadratic residue modulo  $p$  but not  $q$ ,  $jk_p$  is a quadratic residue but  $jk_q$  is not;  $-jk_q$  is the quadratic residue instead.

Then the decryption oracle outputs the numbers corresponding to  $(k_p, k_q)$  and  $(jk_p, -jk_q)$ . The Chinese remainder theorem tells us that these numbers are  $pk_q + qk_p$  and  $jqk_p - jpk_q$ . We can divide the second one by  $j$  to get a different root of  $k^2$ , namely  $qk_p - pk_q$ . But now the attacker can add the two roots together to get  $2qk_p$ . If he takes the greatest common factor of this sum and  $N$ , he'll get  $q$ , one of the factors, and can find  $p$  from that. Then, knowing  $p$  and  $q$ , the attacker will be able to decrypt messages just as well as the intended recipient.

Since about half the numbers modulo a prime are quadratic residues, and about half aren't, a randomly chosen  $j$  will be a square modulo  $p$  but not  $q$  (or vice versa) about half the time. This is really bad. There are, however, ways to protect against such attacks.

This encryption scheme takes  $O(k^2)$  time to encrypt a message and  $O(k^3)$  time to decrypt, where  $k$  is the length of the secret key  $(p, q)$ . This is about the same as RSA.

## 5 Single-bit encryption with nondeterministic padding

In our analyses of cryptosystems thus far, we've assumed that the messages were to be chosen randomly, and proved that our cryptosystem was hard to crack. But what if the system is easy to crack over the actual message space?

Regardless, we can probably assume that there are approximately the same number of 0's and 1's if we look at our messages bit by bit. We can randomly pad each bit into a longer string and then encode the string. The added randomness allows a 0 or 1 to be mapped to one of many strings, so most of the encoded bits look different, even though there are really only two possibilities.

There are two things that we need for this to work: First, we want to be able to undo the random padding to get our original bit back with the aid of a secret key. Second, we want the bit that we care about (as opposed to all the padding) to be secure against any attacks.

The idea of a string being difficult to decode in general but easy with a secret key suggests that we use a trapdoor function. In general, we can encrypt a single bit in the manner described with any trapdoor function as follows:

- Key generation - Pick a trapdoor function  $f$ , with secret  $t_f$ , and pick a hard-core predicate  $B$  for  $f$ .
- Encryption - To encrypt a bit  $m$ , randomly select  $x$  such that  $B(x) = m$ , and output  $f(x) = c$ .
- Decryption - Find  $x = f^{-1}(c)$  with the help of the secret key, and return  $B(x) = m$ .

This encryption scheme divides the image of  $f$  into strings that correspond to 0 and strings that correspond to 1. The attacker cannot distinguish which category some  $f(x)$  falls into because  $B$  is a hard-core predicate.

## 6 Trapdoor predicates

These conditions, that  $f$  be a trapdoor function and  $B$  be a hardcore predicate, are stronger than they need to be. All that our single-bit encryption scheme really needs to work is that an attacker cannot distinguish between strings that correspond to 0 and those that correspond to 1, but we can. Also, the scheme would be rather useless if there was no efficient way to encode bits.

This is exactly the definition of a *trapdoor predicate*. More formally, a trapdoor predicate is a boolean function  $B : \{0, 1\}^* \Rightarrow \{0, 1\}$  that satisfies the following:

- We need to encrypt efficiently - There is a probabilistic polynomial-time algorithm  $\mathcal{A}$  that outputs random  $x$  such that  $B(x) = m$ , the bit we are trying to encode.
- We need our bit to be secure - For any probabilistic polynomial-time algorithm  $\mathcal{B}$ , we want  $\Pr[\mathcal{B}(x) = B(x)] < \frac{1}{2} + p(k)$  for any non-negligible function  $p$ .
- We need to be able to decode - Given some secret key  $s$ , we need to be able to compute  $B(x)$  in probabilistic polynomial-time.

Now let's look at an example of a trapdoor predicate. We'll use a scheme similar to the squaring scheme given above, based on a similar problem of trying to distinguish quadratic residues from nonresidues.

- Key generation - Generate two large primes  $p$  and  $q$  such that  $p \cong q \cong 3 \pmod{4}$ . Keep  $(p, q)$  as the secret key, but announce the public key  $N = pq$ . (Just like before.)
- Encryption - For our message bit  $m$ , randomly generate  $x \in \mathbb{Z}_N^*$  and output  $\text{Enc}(N, m) = (-1)^m x^2 = c$ . Now since -1 is a nonresidue,  $c$  is a quadratic residue if and only if  $m = 0$ .
- Decryption - Decide whether  $c$  is a quadratic residue, and return 0 if it is and 1 if it's not.

We can use the same algorithm as before to determine whether  $c$  is a quadratic residue or not, if we know  $p$  and  $q$ : just try to take the square root and if there aren't any, it's not a perfect square. But if we don't know  $p$  and  $q$ , figuring out whether  $c$  is a quadratic residue or not is just as hard as actually finding the roots. And earlier we realized that this is as hard as factoring  $N$ , which is a very hard problem!

## 7 Using single-bit encryption for arbitrary length encryption

Now that we can encrypt single bits, can we encrypt a multibit message bit-by-bit and expect security? The answer is yes!

Let  $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$  be a semantically secure probabilistic encryption scheme for single bits. Then define *probabilistic encryption*  $PE_\Pi = (\text{G}, \text{E}, \text{D})$  as follows:

- $G(1^k) = \text{Gen}(1^k)$
- $E(pk, m = m_1 \cdots m_l) = \text{Enc}(pk, m_1) \cdots \text{Enc}(pk, m_l)$ : bit-by-bit encryption
- $D(sk, c = c_1 \cdots c_l) = \text{Dec}(sk, c_1) \cdots \text{Dec}(sk, c_l)$ : bit-by-bit decryption

**Theorem 1.** *If  $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$  is semantically secure for single bits, then  $PE_\Pi$  is semantically secure for messages of any length.*

Proof: The proof is by hybridization. Suppose there existed a probabilistic polynomial time adversary algorithm  $\mathcal{A}$  and a message space  $\mathcal{M}$  with  $l$  bit messages, such that over a random choice of two messages  $m_0$  and  $m_1$  from  $\mathcal{M}$  and a random encryption of one of those messages  $c$ ,

$$P[\mathcal{A}(e, m_0, m_1, c) = m] > \frac{1}{2} + \frac{1}{P(k)} \quad (2)$$

for infinitely many  $k$  and some polynomial  $P$ . (That is, the adversary can frequently guess with nonnegligible probability which of two messages has been encrypted.)

Then we can use this adversary to decrypt, with non-negligible probability, single encrypted bits as follows.

The existence of  $\mathcal{A}$  implies that for infinitely many message lengths  $k$ ,  $\mathcal{A}$  can distinguish between encrypted random  $m_0^k, m_1^k \in \mathcal{M}$  with better than negligible probability over (a) choice of encryption keys and (b) the probabilistic ciphertexts  $c_1 = E(e, m_1^k), c_2 = E(e, m_2^k)$ .

Now pick a  $k$  such that  $\mathcal{A}$  can distinguish messages generated from the space  $\mathcal{M}(1^k)$  well:

$$P[A(e, c_0) = 1] - P[A(e, c_1) = 1] > \alpha \text{ over} \quad (3)$$

$$(e, d) \in G(1^k), \quad (4)$$

$$c_0 \in E(e, m_0^k), \quad (5)$$

$$c_1 \in E(e, m_1^k). \quad (6)$$

We are now ready to receive our encrypted bit  $c$ , encrypted with public key  $e$ .

1. Choose two messages,  $m_0^k, m_1^k$ , of length  $l$ , whose ciphertexts  $\mathcal{A}$  can distinguish. Without loss of generality, assume that  $\mathcal{A}$  outputs 1 more often on message  $m_1^k$  than on  $m_0^k$ .
2. Construct a series of hybrid messages  $s_i$ , where  $s_i$  consists of the first  $i$  bits of  $m_0^k$  and the last  $l - i$  bits of  $m_1^k$ . We know that

$$\sum_{i=1}^l (P[A(e, \text{Enc}(e, s_{i+1})) = 1] - P[A(e, \text{Enc}(e, s_i)) = 1]) > \alpha \quad (7)$$

Therefore, there must be some particular  $i$  such that

$$P[A(e, \text{Enc}(e, s_{i+1})) = 1] - P[A(e, \text{Enc}(e, s_i)) = 1] > \frac{\alpha}{l} \quad (8)$$

Generate a ciphertext  $(c_1, \dots, c_l) = \text{Enc}(e, s_i)$  and stick our mystery encrypted bit  $c$  in position  $i$ . Then compute  $b = A(c_1, \dots, c, \dots, c_l)$  and output the  $i$ th bit of  $m_1^k$  if  $b = 1$  or the  $i$ th bit of  $m_0^k$  if  $b = 0$ .

If  $\mathcal{A}$  exists, then the above algorithm decrypts single bits with nonnegligible probability, which contradicts our assumption that our single-bit probabilistic scheme achieved semantic security. Therefore, the multi-bit scheme is semantically secure also.

## 8 Homomorphic encryption

**Definition 2.** An encryption scheme is **homomorphic** if computations on the ciphertexts are reflected as computations on the messages when decrypted.

Symbollically, given an encryption function  $E$  and messages  $m_1$  and  $m_2$ , this means

$$E(m_1) \circ E(m_2) = E(m_1 \diamond m_2).$$

Note that the operation on the cyphertexts and the messages can be different operations. Example applications of homomorphic encryption schemes:

- E-voting: all votes could be encrypted and include a 0 or 1 indicating the vote. The ciphertexts of the votes could be added and then decrypted, yielding the vote count without revealing individual votes.
- Secure cloud computing: data could be encrypted and have ciphertexts operated on (both  $+$  and  $\times$ ) without revealing the data itself. The result would then be decrypted and used.

Specifically with the quadratic residuosity encryption scheme, we have the following homomorphic properties:

- $E(m_1 \oplus m_2) = E(m_1) \cdot E(m_2)$  (can be checked with truth table)
- $E(1 \oplus m) = E(1) - E(m)$
- $E(m) = E(0) \cdot E(b)$  (effectively re-randomizing)

## 9 Probabilistic encryption scheme and examples

### 9.1 Main Idea

Given a trapdoor permutation collection  $F$ , we define an encryption scheme as follows:

- The key generation function  $\text{Gen}(1^k)$  simply chooses a  $f \in F$  and its corresponding trapdoor  $t$ , and outputs  $(f, t)$ .
- The encryption function  $\text{Enc}(f, m)$  chooses a seed  $r$  in the domain of  $f$  and a PSRG  $g$  based on  $f$ . It returns  $c = (c_1, c_2) = (g(r) \oplus m, f^{|m|+1}(r))$ .
- The decryption function  $\text{Dec}(t, (c_1, c_2))$  has access to the trapdoor. It first finds  $r = f^{-(|m|+1)}(c_2)$  (by inverting over and over) then returns  $m = c_1 \oplus g(r)$ .

The security of this scheme follows from the assumption of a PSRG.

### 9.2 Example: RSA

The probabilistic approach can be applied to RSA as follows:

- $\text{Gen}(1^k)$  is defined as choosing  $(n, e)$  just as in RSA.
- $\text{Enc}(n, m)$  is defined by choosing  $r \in Z_n^*$  and concatenating  $|m|$  bits computed by

$$\text{pad} = \text{lsb}(r \bmod n) \quad \text{lsb}(r^e \bmod n) \quad \text{lsb}(r^{e^2} \bmod n) \quad \dots \quad \text{lsb}(r^{e^{|m|-1}} \bmod n).$$

We then set  $c = (\text{pad} \oplus m, r^{|m|})$ .

- $\text{Dec}((p, q), (c_1, c_2))$  decrypts by finding  $r$  as the  $|c_1|$ th root of  $c_2$  modulo  $n$  (using the factorization  $n = pq$ ). Then, it can recompute pad as above and find  $m = c_1 \oplus \text{pad}$ .

### 9.3 Example: El Gamal

The El Gamal Cryptosystem is based on the discrete log problem and takes advantage of probabilistic encryption, defined as follows:

- $\text{Gen}(1^k)$  chooses a random  $k$ -bit prime  $p$  such that  $p = 2q + 1$ , where  $q$  is also prime. Let  $g$  be a generator of  $QR_p$ ,  $x$  be a number with  $1 < x < q$ , and  $y = g^x \mod p$ . Publish  $(p, g, y)$  as the public key and keep the  $x$  that was used secret.
- $\text{Enc}((p, g, y), m)$  (where  $m \in QR_p$ ) is defined by choosing randomly  $1 \leq r \leq q$ , computing  $\text{pad} = y^r = g^{xr} \mod p$ , and yielding  $c = (\text{pad} \cdot m \mod p, g^r)$ .
- $\text{Dec}(x, (c_1, c_2))$  is able to decrypt the cipher by recomputing the pad as  $\text{pad} = c_2^x = g^{rx} \mod p$  and finding  $m = c_1 \cdot \text{pad}^{-1} \mod p$ . by finding  $r$  as the  $|c_1|$ th root of  $c_2$  modulo  $n$  (using the factorization  $n = pq$ ). Then, it can recompute pad as above and find  $m = c_1 \oplus \text{pad}$ .

Note that  $g$  and  $p$  can be shared across all the users as long as  $x$  and therefore  $y$  are chosen differently for each key generation.

This scheme has, for message size  $|m| = k$ , public key of size  $O(k)$ , bandwidth of  $O(k)$ , and both encryption and decryption running time of  $O(k^3)$ . We also have security:

**Theorem 3.** *Under DDH, El Gamal is computationally indistinguishable.*

El Gamal also has multiplicative homomorphism. That is, if  $\text{Enc}(m) = (c_1, c_2)$  and  $\text{Enc}(m') = (c'_1, c'_2)$ , we have  $\text{Enc}(m \cdot m') = (c_1 c'_1 \mod p, c_2 c'_2 \mod p)$ .

Furthermore, we can modify the scheme to also have additive homomorphism as follows. In encrypting, instead of returning  $c_1 = \text{pad} \cdot m \mod p$ , we set  $c_1 = \text{pad} \cdot g^m \mod p$ . With this modification, multiplying  $g^m \cdot g^{m'} = g^{m+m'}$  effectively adds  $m + m'$ . To decrypt, as long as  $m$  is a member of a polynomial size known set, can try all possibilities for  $g^m$  and choose the one that matches.

### 9.4 Example: Paillier

Another example of an encryption scheme that uses randomness is as follows:

- $\text{Gen}(1^k)$  chooses a  $n = pq$ , where  $p$  and  $q$  are primes. It publishes  $n$  and keeps  $\phi(n)$  secret.
- $\text{Enc}(n, m)$  (assuming  $m \in Z_n^*$ ) chooses a random  $r \in Z_n^*$  and computes

$$c = (1 + n)^m r^n \mod n^2.$$

- $\text{Dec}((p, q), c)$  first computes

$$\begin{aligned} c' &= c^{\phi(n)} \mod n^2 \\ &= (1 + n)^{m\phi(n)} r^{n\phi(n)} \mod n^2 \\ &= (1 + n)^{m\phi(n)} \mod n^2 \\ &= 1 + nm\phi(n) \mod n^2, \end{aligned}$$

from which we can find  $m = \frac{c' - 1}{n\phi(n)}$ .

Note that the last step of decryption follows from the fact  $(1 + n)^t = 1 + tn + n^2(\dots) = 1 + tn \mod n^2$  for any  $t$ .

The Paillier encryption scheme is used in applications such as auctions and voting due to its homomorphic properties: if  $\text{Enc}(n, m) = c$  and  $\text{Enc}(n, m') = c'$ , then  $\text{Enc}(n, m + m' \mod n) = c \cdot c'$  and  $\text{Enc}(n, m - m' \mod n) = c/c'$ .

The security of the scheme is guaranteed under the Decisional Composite Residuosity (DCR) assumption.

**Definition 4.** *The Decisional Composite Residuosity (DCR) assumption states that it is hard to distinguish between  $(n, R^n)$  and  $(n, S)$  for random  $R \in Z_n$  and  $S \in Z_{n^2}$ .*

With DCR, Paillier is computationally indistinguishable against a passive adversary.