6.875J/18.425J Cryptography and Cryptanalysis Mar 9, 2016

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 and incorporate randomness before encrypting it with RSA. One scheme is Optimal Asymmetric Encryption Padding (OAEP) [**?**]:

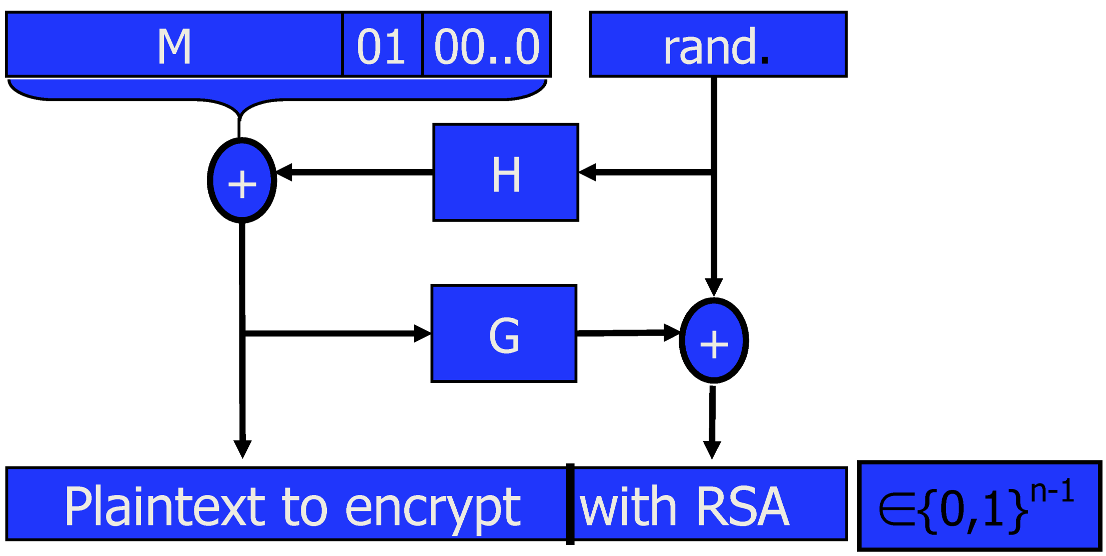

Figure 1: The OAEP block diagram

RSA with OAEP is semantically secure when H, G are *random oracles*.

Intuitively, 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. In the proofs below, we use an equivalent formulation of semantic security called computational indistinguishability, where the attacker is allowed to pick her favorite two messages and told that the ciphertext is one of them encrypted. Despite that information, she still cannot distinguish which message the ciphertext decrypts to.

2 Random Oracles in Theory and Practice

A random oracle H on inputs x produces outputs $H(x)$ that are truly random but consistent. That is, if queried multiple times with the same input x, the oracle always produces the same output. We model a random oracle as a function chosen truly randomly from the set of all possible functions mapping $\{0,1\}^* \to \{0,1\}^l$, 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, Goldreigh, and Halevi exhibited a crypto-system proven secure under the random oracle model but insecure with any particular implementation of the oracle [**?**].

3 Implementation Attacks on RSA

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

Because vanilla RSA (without OAEP) is deterministic, it is insecure against adversaries who can obtain encryptions of arbitrary messages (i.e. CPA and CCA adversaries).[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. It's easy to accidentally make an insecure implementation, so don't roll your own cryptography!

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 believed to be hard to find the square root of a given perfect square. (Perfect squares are also called quadratic residues.) However, it is easy 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 $p \equiv q \equiv 3 \pmod{4}$. The secret key is (p,q) and the public key is $N = pq$.
- Encryption - $\mathsf{Enc}(N,m) = c = m^2 \pmod{N}$
- Decryption - $\mathsf{Dec}(N,c)$ gives the unique square root of c that is itself a perfect square modulo N.

[1] 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. A chosen-plaintext attack is more limited: Mallory can request encryptions of arbitrary messages, but she cannot decrypt arbitrary messages.

We can formalize the chosen-ciphertext attack as

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

.

The message space M is the set of quadratic residues modulo N. (We can encrypt a more general message x by choosing N very large and taking encoding $m = x^2$.)

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 CCA1 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?

We can break the message into individual bits and encrypt it bit-by-bit, randomly padding each bit into a longer string before encoding. 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. Breaking up the messages allows us to use the sample space of single bits.

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.

We will construct encryption for single bits from trapdoor functions, 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 a 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 sample efficiently - There is a probabilistic polynomial-time algorithm on each input $m \in \{0,1\}$ that outputs random x such that $B(x) = m$.
- We need our bit to be secure - For any probabilistic polynomial-time algorithm $\mathcal{A}$, we want $\Pr[\mathcal{A}(x) = B(x)] < \frac{1}{2} + p(k)$ for any non-negligible function p.
- We need to be able to decode with the trapdoor - 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 the hardness of distinguishing quadratic residues from nonresidues of Jacobi symbol 1.

- Key generation - Generate two large primes p and q such that $p \equiv q \equiv 3 \pmod 4$. 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 $\mathsf{Enc}(N, m) = (-1)^m x^2 = c$. 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 $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a semantically secure probabilistic encryption scheme for single bits. Then define *probabilistic encryption* $PE_\Pi = (\mathsf{G}, \mathsf{E}, \mathsf{D})$ as follows:

- $G(1^k) = Gen(1^k)$
- $E(pk, m = m_1 \cdots m_l) = Enc(pk, m_1) \cdots Enc(pk, m_l)$: bit-by-bit encryption

- $D(sk, c = c_1 \cdots c_l) = Dec(sk, c_1) \cdots Dec(sk, c_l)$: bit-by-bit decryption

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

Proof. The proof is by hybridization. Suppose there existed a probabilistic polynomial time adversary algorithm $\mathcal{A}$ over a message space $\mathcal{M}$ with l bit messages, and a polynomial P. The adversary chooses two messages m_0 and m_1 on input 1^k from $\mathcal{M}$. The challenger then chooses one of the two messages at random and encrypts it to ciphertext c. Suppose the adversary's performance is

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

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

Then we can use $\mathcal{A}$ to construct a second adversary, $\mathcal{B}$, to decrypt, with non-negligible probability, single encrypted bits, as follows. This violates the security guarantee of the single-bit scheme.

The existence of $\mathcal{A}$ implies that for infinitely many values of the security parameter k, $\mathcal{A}$ can distinguish between encrypted $m_0^k, m_1^k \in \mathcal{M}$ with better than negligible probability over (a) the adversary's choice of m_0^k, m_1^k, (b) the choice of encryption keys and (c) the encryption randomness used to generate ciphertexts $c_1 = E(pk, m_1^k), c_2 = E(pk, m_2^k)$. Pick a k such that $\mathcal{A}$ can distinguish messages generated from the space $\mathcal{M}(1^k)$ well:

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

$$(pk, sk) \in G(1^k), \tag{4}$$

$$c_0 \in E(pk, m_0^k), \tag{5}$$

$$c_1 \in E(pk, m_1^k). \tag{6}$$

Now construct adversary $\mathcal{B}(pk, c)$ as follows:

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 A outputs 1 with greater probability 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

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

   Therefore, there must be some particular i such that

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

   Generate a ciphertext $(c_1, \cdots, c_l) = Enc(pk, s_i)$ and stick our challenge encrypted bit c in position i. Then compute $b = A(c_1, \cdots, c, \cdots, c_l)$ and output the ith bit of m_1^k if $b = 1$ or the ith 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 include:

- 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)
- $E(1 \oplus m) = E(1) - E(m)$
- $E(m) = E(0) \cdot E(m)$ (effectively re-randomizing)

9 Probabilistic encryption scheme and examples

9.1 Main Idea

Bit-by-bit encryption increases ciphertext size, computation cost, and randomness used. We can improve on efficiency while preserving semantic security using the following efficient probabilistic encryption scheme. Given a trapdoor permutation collection F, we define the encryption scheme as follows:

- The key generation function $\mathsf{Gen}(1^k)$ simply chooses a $f \in F$ and its corresponding trapdoor t, and outputs (f, t).
- The encryption function $\mathsf{Enc}(f, m)$ 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 $\mathsf{Dec}(t, (c_1, c_2))$ 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 efficient probabilistic encryption approach can be applied to RSA as follows:

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

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

  We then set $c = (\text{pad} \oplus m, r^{|m|})$.
- $\mathsf{Dec}((p, q), (c_1, c_2))$ 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:

- $\mathsf{Gen}(1^k)$ 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 $y = g^x \mod p$. Let (p, g, y) be the public key and x be the secret key.
- $\mathsf{Enc}((p, g, y), m)$ (where $m \in QR_p$) is defined by choosing randomly $1 \leq r \leq q$, computing pad $= y^r = g^{xr} \mod p$, and yielding $c = (\text{pad} \cdot m \mod p, g^r)$.
- $\mathsf{Dec}(x, (c_1, c_2))$ is able to decrypt the cipher by recomputing the pad as 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 (i.e. CPA secure).*

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

We can modify the scheme to 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:

- $\mathsf{Gen}(1^k)$ chooses a $n = pq$, where p and q are primes. It publishes n and keeps $\phi(n)$ secret.
- $\mathsf{Enc}(n, m)$ (assuimg $m \in Z_n^*$) chooses a random $r \in Z_n^*$ and computes
$$c = (1+n)^m r^n \mod n^2.$$
- $\mathsf{Dec}((p, q), c)$ 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 $(1+n)^t = 1 + tn + n^2(\cdots) = 1 + tn \mod n^2$ for any t.

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

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.

References

[BG84] Michael Blum and Shafi Goldwasser. An efficient probabilistic public-key encryption scheme which hides all partial information. *Advances in Cryptology - Crypto 84*, 196:289–302, 1984.

[BR95] Mihir Bellare and Phillip Rogaway. *Advances in Cryptology — EUROCRYPT'94: Workshop on the Theory and Application of Cryptographic Techniques Perugia, Italy, May 9–12, 1994 Proceedings*, chapter Optimal asymmetric encryption, pages 92–111. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *Journal of the ACM*, 51(4):0 – 594, 2004.

[Cip03] M. Cipolla. Un metodo per la risoluzione della congruenza di secondo grado. *Rendiconto dellAccademia Scienze Fisiche e Matematiche*, IX:154–163, 1903.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 1985.