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*

1 RSA preprocessing

2 Random Oracle Model Methodology

3 Types of Attacks

- Lunchtime attack
- Timing attack
- Power attack
- Fault attack
- Cache attack

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 - $\mathsf{Enc}(N, m) = c = m^2 \pmod N$
- Decryption - $\mathsf{Dec}(N, c)$ gives the unique square root of c that is itself a perfect square modulo N.

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 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 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 $\mathsf{Enc}(N, m) = (-1)^m x^2 = c$. 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

8 Homomorphic encryption

Definition 1. *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 $\mathsf{Gen}(1^k)$ 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 probabilistic 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 $r \in Z_n^*$ 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 $y = g^x \mod p$. Publish (p, g, y) as the public key and keep the x that was used secret.
- $\mathsf{Enc}((p, g, y), m)$ (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 $c = (\text{pad} \cdot m \mod p, g^r)$.
- $\mathsf{Dec}(x, (c_1, c_2))$ 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 2. *Under DDH, El Gamal is computationally indistinguishable.*

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 $\mathsf{Enc}(m \cdot m') = (c_1 c_1' \mod p, c_2 c_2' \mod p)$.

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:

- $\mathsf{Gen}(1^k)$ 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 3. *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