Lecture 15: Secure Public Key Encryption

1 Square roots modulo n are as hard as factoring

Recall from the previous lecture that a trapdoor one-way permutation is one of the core primitives in building a public key encryption algorithm. A trapdoor one-way permutation is a pair of functions (f, f^{-1}) such that f can be computed in polynomial time, but its inverse can not, without the knowledge of the trapdoor. The Rabin scheme used the fact that squaring modulo an RSA modulus is a trapdoor one-way permutation. To prove this fact, we begin by proving that squaring modulo n is hard to invert.

Theorem 1 *Assuming factoring is hard, if n is an RSA modulus then*

$$\mathop{\mathbf{Pr}}_{x \xleftarrow{\$} (\mathbb{Z}/n\mathbb{Z})^*} \left[A(n, x^2 \mod n) = x\right] \tag{1}$$

is negligible for all efficient algorithms A.

Proof: (by contrapositive)
Suppose the probability in equation (1) is large. We can then use algorithm A to factor n. Consider another probability, which is at least as large:

$$\mathop{\mathbf{Pr}}_{x \xleftarrow{\$} (\mathbb{Z}/n\mathbb{Z})^*} \left[A(n, x^2 \mod n)^2 = x^2\right] = p$$

The first probability is that A outputs a particular square root of its input. The second probability, p, just requires that A output any square root of its input. We will see that if p is large then n can be factored efficiently, which is stronger than the theorem statement.

First, remember that squaring modulo n is a 4-to-1 function. Any number $x^2 \mod n$ has 4 possible square roots, which we know to be x, $-x$, ωx, and $-\omega x$, where ω is a nontrivial square root of one. Thus the event that A succeeds in outputting a square root can be partitioned into 4 groups, leading to the following:

$$\begin{aligned}
\mathbf{Pr}\left[A(n, x^2 \mod n) = x\right] &= p/4 \\
\mathbf{Pr}\left[A(n, x^2 \mod n) = -x\right] &= p/4 \\
\mathbf{Pr}\left[A(n, x^2 \mod n) = \omega x\right] &= p/4 \\
\mathbf{Pr}\left[A(n, x^2 \mod n) = -\omega x\right] &= p/4
\end{aligned}$$

These probabilities must all be equal—if they were not, A would effectively be distinguishing whether one root was squared or another. But there is no way for A to differentiate; it sees only the square, which is the same for any of the 4 roots. The probabilities sum to p, so each has value $p/4$.

Using this information, we can construct the algorithm B, which efficiently factors n:

B (n)
$x \xleftarrow{\$} (\mathbb{Z}/n\mathbb{Z})^*$
$y \leftarrow x^2 \mod n$
$z \leftarrow A(n, y)$
if $z \neq \pm x$, output $\gcd(x - z, n)$, $\gcd(x + z, n)$

If A outputs a value for z other than x or $-x$, B succeeds in factoring n. For any value of x, there are 2 values of z that would work, so the probability of getting one of them is $p/2$. Thus the probability B factors n is $p/2$. Running B several times as Bernoulli trials, we expect to succeed in $O(1/p)$ tries, so the running time to factor n is $O(1/p \cdot \text{runtime of} A)$. This algorithm factors n efficiently unless p is very small, and so if we assume factoring is hard we can conclude that finding square roots modulo n is also hard. $\square$

Corollary 2 *$f : x \longmapsto x^2 \mod n$ is a trapdoor one-way permutation if factoring is hard.*

We already know it is easy to invert squaring if ω is known. The theorem proved it is hard to invert squaring otherwise. So provided that x and n are chosen such that f is in fact a permutation, f is a trapdoor one-way permutation. The conditions are easy to satisfy—n must be a Blum integer (a product of 2 primes each congruent to 3 $(\mod 4)$), and x must be a perfect square, which can be sampled randomly by squaring a random value from $(\mathbb{Z}/n\mathbb{Z})^*$

A simple encryption scheme one might think of is the following:

$$E(m) = (f(x), x + m \mod n)$$
$$D(y, z) = z - f^{-1}(y) \mod n$$

where x is chosen randomly for each encryption. However, this scheme does not ensure indistinguishability over plain-text attacks. The security notion of f, as a trapdoor one-way permutation, ensures that x can not be computed from $f(x)$. But it does not rule out the possibility that an adversary might be able to compute some part of x easily, and knowing part of x reveals part of m.

It turns out some trapdoor one-way permutations do have the property that it is hard to compute any bit of x given $f(x)$. The key concept involved in the proof is the notion of "hard core bits."

2 Hard core bits

Before formally defining a "hard core bit," an example provides some intuition. Say the trapdoor one-way permutation is $f(x) = x^3 \mod n$. Now consider what happens if it is possible to compute the least significant bit of x.

Claim 3 *If $A(n, f(x)) = lsb(x)$, then f^{-1} can be computed efficiently.*

The notation $[x] = f(x)$ will indicate that $f(x)$ gives some kind of a concealed version of $[x]$. The value $[x]$ does not reveal what x is, but one can do certain manipulations on it. For example, given the value $[x]$, one can compute $[2x] = f(2x \mod n)$. Since f is the cube $\mod n$, $f(2x \mod n)$ is simply $2^3x^3 \mod n = 8x^3 \mod n = 8 \cdot f(x) \mod n$.

Think of x as a string of bits, and A has given us the least significant bit (suppose it was 0). So we have $[x]$, as shown below, and we can compute $[2x]$. Notice that doubling x shifts all the bits to the left by 1 and appends a 0 to the end. So we know the last 2 bits of x are 00. But what if doubling causes the value to be greater than n? Then n is subtracted from the result and the last 2 bits may no longer be 00. However, n is public so we know its value. Moreover, we know n is odd, so if it is subtracted then the least significant bit will flip. Because our algorithm A can compute the least significant bit, it can detect if n has been subtracted. Suppose n is not subtracted after doubling. So we try doubling again, which will shift the bits once more to the left. But this time n is subtracted. We expected the last 3 bits of $4x$ to be 000, but after subtracting n they will have a new value. Since we know n we know what that new value will be (in this case, say it is 101).

$$
\begin{array}{ll}
x: & [\cdots\cdots\cdots\cdots\cdots\cdots\cdots 0] = [x] = f(x) \\
2x \mod n: & [\cdots\cdots\cdots\cdots\cdots\cdots\cdots 00] \\
4x \mod n: & [\cdots\cdots\cdots\cdots\cdots\cdots 000] \rightarrow [\cdots\cdots\cdots\cdots\cdots\cdots 101]
\end{array}
$$

We can go on doubling, and at each step learn one more bit of the result. On step k we know the last $k-1$ bits of $2^{k-1}x \mod n$. After $\log n$ steps, we know the complete value of $2^{\log n}x \mod n$, from which we can compute x. Starting from $f(x)$, we were able to compute x using only $\log n$ invocations of A. This contradicts the RSA assumption that cubing modulo n is a trapdoor one-way permutation, which proves that no function A exists which always computes $\mathrm{lsb}(x)$ given only n and $f(x)$. A similar property holds for the Rabin scheme, where $f(x) = x^2 \mod n$.

Now we can devise a new algorithm. If we limit the message length to 1 bit, the encryption algorithm can be

$$E(m) = (f(x), m \oplus \mathrm{lsb}(x))$$

where f is one of the functions that is known to conceal its least significant bit. Is this algorithm secure against plain text attacks? Unfortunately, the property we have proven is still not enough. It does not cover algorithms that can predict the least significant bit with less than 100% probability. However, it is possible to prove a stronger claim:

Claim 4 *If lsb(x) can be predicted from $x^3 \mod n$ in time t with probability $\frac{1}{2} + \frac{\epsilon}{2}$, then n can be factored in time $O(t(logn)^2/\epsilon^2)$.*

The formal proof of the claim will not be shown here, but the intuition is that more invocations of A reveal more about $\mathrm{lsb}(x)$. A is called on to compute $\mathrm{lsb}(rx \mod n)$ for many random values of r, and if the correctness probability of A is more than $\frac{1}{2}$, then it will almost always be able to accurately determine $\mathrm{lsb}(x)$.

We now have the sense that the least significant bit of x is a hard bit to predict. This leads to the definition of a hard core bit:

Definition 5 *if $f: X \longrightarrow X$ is a one way permutation and $B: X \longrightarrow \{0,1\}$ is a predicate, B is a (t,ϵ)-hard core bit for f if*

$$\mathsf{Adv}\ A \le 2\left|\mathbf{Pr}\left[x \xleftarrow{\$} X; A(f(x)) = B(x)\right] - \frac{1}{2}\right| \quad \forall A \text{ running in time} \le t$$

3 Goldwasser-Micali

Goldwasser-Micali is an encryption algorithm that makes use of a hard core bit. It is a generalization of the previous encryption algorithm:

$$E(m) = (f(x), B(x) \oplus m)$$

where $x \xleftarrow{\$} X$ and $f(x)$ is a trapdoor one-way permutation, and B is a hard core bit.
A hard core bit can also be used to construct a PRG, whose output is one longer than its input:

$$\begin{aligned} &G : \{0,1\}^k \longrightarrow \{0,1\}^{k+1} \\ &G(x) = (f(x), B(x)) \end{aligned}$$

G works as long as f is a one-way permutation and B is a hard core bit.

Theorem 6 *If B is a (t, ϵ)-hard core bit for (F, F^{-1}), then Goldwasser-Micali is $(t - O(1), 1, \epsilon)$-IND-CPA*

PROOF: Suppose not, so A breaks Goldwasser-Micali:

$$\mathsf{Adv}\ A = \left|\mathbf{Pr}\left[A^{E_{pk}(\cdot)}(pk) = 1\right] - \mathbf{Pr}\left[A^{E_{pk}(\$(\cdot))}(pk) = 1\right]\right| > \epsilon$$

$$\begin{aligned} \text{so } &\mathbf{Pr}\left[A'^{E_{pk}(\cdot)}(pk) = m\right] > \frac{1}{2} + \frac{\epsilon}{2} \\ &\mathbf{Pr}\left[A''(pk, E_{pk}(m)) = m\right] > \frac{1}{2} + \frac{\epsilon}{2} \\ &\mathbf{Pr}\left[A'''(pk, F_{pk}(x), B(x) \oplus m) = m\right] > \frac{1}{2} + \frac{\epsilon}{2} \end{aligned}$$

and finally, labeling $B(x) \oplus m$ as z, we have

$$\mathbf{Pr}\left[A'''(pk, F_{pk}(x), z) = z \oplus B(x)\right] > \frac{1}{2} + \frac{\epsilon}{2}$$

The three inputs to A''' are all uniformly random, but the output favors a function of the hardcore bit of $f(x)$, without knowing x.

We can now define a function P as

```
P (pk, y)
    pick z <-$- {0,1}*
    output A'''(pk, y, z)
```

P can be usd to predict the hardcore bit:

$$\mathbf{Pr}\left[P(pk, F_{pk}(x)) = B(x)\right] = \mathbf{Pr}\left[A'''(pk, F_{pk}(x), z) = z \oplus B(x)\right] > \frac{1}{2} + \frac{\epsilon}{2}$$

So if there is an adversary that can break Goldwasser-Micali using one chosen plain text oracle query, then there is an other adversary that can predict the hardcore bit. $\square$

Theorem 7 *If (E, D) is a public-key encryption scheme that is $(t, 1, \epsilon)$-IND-CPA, then it is $(t, q, q\epsilon)$-IND-CPA*

PROOF: Consider the case where $q = 2$. An adversary A is trying to distinguish between the real world and the ideal world. In the real world, it makes 2 queries (m and m') to the encryption oracle, so it gets the values $(E_{pk}(m), E_{pk}(m'))$, and then does some calculation. In the ideal world, it gets $(E_{pk}(\$(m)), E_{pk}(\$(m')))$ from its oracle. The real and ideal world can be shown to be indistinguishable by the use of a hybrid. The hybrid returns $(E_{pk}(m), E_{pk}(\$(m')))$ to the algorithm. The real world scenario is at most ϵ-distinguishable from the hybrid—if it were more, another algorithm B would distinguish between the real and random world in one query. B would simply compute $E_{pk}(m)$ on its own, which it can do since pk is public, and then query its oracle on m'. It would use those two values the same way A did, and output the correct world with better than ϵ probability. The hybrid is at most ϵ-distinguishable from the ideal world by a similar argument. An algorithm B' making a single query could query its oracle on m, then randomize m' on its own and compute the encryption of it. So with 2 oracle queries, the scheme is $(t, 2, 2\epsilon)$-IND-CPA. For more queries, a longer chain of hybrids could be used, each adding a factor of ϵ. $\square$