University of Maryland
CMSC456 — Introduction to Cryptography
Professor Jonathan Katz

Lecture 8

1 A One-Way Permutation Equivalent to Factoring

Recall the theorem we wanted to prove from last time:

Theorem 1 *(Informally:) The squaring permutation is hard to invert (i.e., one-way) if and only if factoring is hard.*

As a historical note, the squaring permutation is also known as the *Rabin* permutation after Michael Rabin, who first noticed (in 1979 — one year after RSA was introduced) that the one-wayness of this function was equivalent to hardness of factoring.

Proof Let's first show the easy direction: that that squaring permutation[1] $f_N : \mathcal{QR}_N \to \mathcal{QR}_N$ *cannot* be hard to invert if factoring is easy. If factoring is easy, we have the following algorithm for inverting f_N: on input $y \in \mathcal{QR}_N$, first find the factors p, q of N. Then compute the Chinese remaindering representation (y_p, y_q) of y. Find a square root x_p of y_p modulo p and a square root x_q of y_q modulo q (recall we mentioned that for p *prime*, it is known how to efficiently compute square roots in $\mathbb{Z}_p^*$). Then (x_p, x_q) is a square root of y (verify this!).

We now turn to the more difficult direction. As we mentioned at the end of the last lecture, we will take any algorithm that can invert f_N and use it to factor N. Since this is our first proof of this sort, we will give some more detail about what it is that we will actually prove. Our aim is to show that if we have an efficient algorithm A that can invert f_N (that is, can compute square roots in $\mathcal{QR}_N$), then we can build another efficient algorithm A' that can factor numbers. Note there are two components to the proof: we need to construct an algorithm A' that factors numbers (given an arbitrary algorithm A that inverts f_N) and we also need to ensure that our construction is efficient; that is, that A' runs in polynomial time (assuming that A runs in polynomial time).

More formally, assume we have an efficient algorithm A such that:

$$\Pr[N \leftarrow \mathsf{CompositeGen}(1^k); y \leftarrow \mathcal{QR}_N; z = y^2; x \leftarrow A(z, N) : x^2 = z] > \epsilon(k), \tag{1}$$

for some (arbitrary) function $\epsilon(k)$. A bit about the notation: recall that "$\leftarrow$" refers to the output of a random process, while "$=$" either means (on the left side of the colon) to assignment or (on the right side of the colon) to equality (i.e., are these two things equal?). In the above notation, $\mathsf{CompositeGen}$ refers to some algorithm which, on input 1^k, outputs a k-bit composite number which is a product of two distinct primes. (Typically, it will be

[1] Note: we assume here that f is a permutation (and thus $N = pq$ where $p = q = 3 \bmod 4$; see the notes for Lecture 7) for convenience only. In fact, hardness of inverting f is equivalent to hardness of factoring even when f is not a permutation.

the product of two $k/2$-bit primes, but the workings of the algorithm are actually irrelevant here.) We stress that the nature of $\mathsf{CompositeGen}$ is irrelevant to the proof of security, so let's for now just assume we have such an algorithm as a black box.

We want to show how to use A to construct an efficient algorithm A' for which:

$$\Pr[N \leftarrow \mathsf{CompositeGen}(1^k); (p,q) \leftarrow A'(N) : pq = N] > \epsilon'(k), \tag{2}$$

where $\epsilon'(k)$ will be related in some way to $\epsilon(k)$. In words: A' will be given a random value N output by $\mathsf{CompositeGen}$, A' will output p, q, and the probability that $pq = N$ (where this probability is taken over the entire experiment) is at least $\epsilon'(k)$. For our proof, we will want it to be the case that if $\epsilon(k)$ is *not* negligible, then $\epsilon'(k)$ will not be negligible either. Hence, if there exists an efficient algorithm A satisfying (1) with $\epsilon(k)$ not negligible, then there exists an efficient algorithm A' satisfying (2) — but this violates the factoring assumption! (Note: whether or not this actually violates the factoring assumption depends on our definition of $\mathsf{CompositeGen}$. So what we are really assuming is that there is some algorithm $\mathsf{CompositeGen}$ for which factoring the output of $\mathsf{CompositeGen}$ is hard.)

Before we present the actual algorithm, we first give the following lemma:

Lemma 1 *Given two elements $x, y \in \mathbb{Z}_N^*$ such that $x^2 = y^2 \bmod N$ and $x \neq \pm y$, one can efficiently factor N.*

The proof of this lemma is quite simple. Given that $x^2 = y^2 \bmod N$, we have:

$$x^2 - y^2 = (x-y)(x+y) = 0 \bmod N,$$

and furthermore $x - y \neq 0 \bmod N$ and $x + y \neq 0 \bmod N$ (why?). Since N divides $(x+y)(x-y)$ but does not divide either of $(x+y)$ or $(x-y)$, it must be the case that $\gcd(x+y, N)$ gives a non-trivial divisor of N (and the gcd can be computed efficiently). Since N is the product of two primes, we are done.

Our algorithm A' will work as follows: On input N, choose a random $y \in \mathbb{Z}_N^*$ and compute $z = y^2 \bmod N$. Run $A(z, N)$ to get output x. if $x^2 = z$ and $x \neq \pm y$, use Lemma 1 to factor N. Otherwise, simply give up.

How can we represent the success probability of our algorithm? Note that, because of Lemma 1, our algorithm succeeds exactly when $x^2 = z$ and $x \neq \pm y$. So,

$$\begin{aligned}
&\Pr[N \leftarrow \mathsf{CompositeGen}(1^k); (p,q) \leftarrow A'(N) : pq = N] = \\
&\quad \Pr[N \leftarrow \mathsf{CompositeGen}(1^k); y \leftarrow \mathbb{Z}_N^*; z = y^2 \bmod N; x = A(z,N) : x^2 = z \wedge x \neq \pm y].
\end{aligned}$$

Let us represent the entire experiment to the left of the colon (in this last expression) by Expt. Conditioning (using basic probability) gives:

$$\Pr[\mathsf{Expt} : x^2 = z \wedge x \neq \pm y] = \Pr[\mathsf{Expt} : x^2 = z] \cdot \Pr[\mathsf{Expt} : x \neq \pm y | x^2 = z].$$

We now make the following two crucial observations. First, we can rewrite $\Pr[\mathsf{Expt} : x^2 = z]$ as follows:

$$\begin{aligned}
&\Pr[\mathsf{Expt} : x^2 = z] \\
&\quad \stackrel{\text{def}}{=} \Pr[N \leftarrow \mathsf{CompositeGen}(1^k); y \leftarrow \mathbb{Z}_N^*; z = y^2 \bmod N; x = A(z,N) : x^2 = z] \\
&\quad = \Pr[N \leftarrow \mathsf{CompositeGen}(1^k); y \leftarrow \mathcal{QR}_N; z = y^2 \bmod N; x = A(z,N) : x^2 = z],
\end{aligned}$$

where this is true because the distribution on z is *exactly* the same regardless of whether we first pick y from $\mathbb{Z}_N^*$ or from $\mathcal{QR}_n$ (why?), and furthermore y does not appear any further in the experiment or in the event itself. Now, *by assumption* we have

$$\Pr[N \leftarrow \mathsf{CompositeGen}(1^k); y \leftarrow \mathcal{QR}_N; z = y^2 \bmod N; x = A(z, N) : x^2 = z] > \epsilon(k)$$

(compare to (1)).

The second observation we make is that $\Pr[\mathsf{Expt} : x \neq \pm y | x^2 = z] = 1/2$. Why is this true? Well, one way to see it is that algorithm A has no idea whatsoever which of the four square roots of z were picked by A'. So no matter what A does, as long as it outputs a value x which is a square root of z, then with probability $1/4$ we will have $x = y$ and with probability $1/4$ we will have $x = -y$. And with probability $1/2$ we will have $x \neq \pm y$.

Putting everything together gives:

$$\Pr[N \leftarrow \mathsf{CompositeGen}(1^k); (p, q) \leftarrow A'(N) : pq = N] > \epsilon(k)/2.$$

In particular, if $\epsilon(k)$ is not negligible, then neither is the success probability of our algorithm A' in factoring N. But this would violate the factoring assumption! So, we conclude that $\epsilon(k)$ must be negligible after all. ■

I presented the above proof in excruciating detail. As you become more familiar with these types of proofs, they become much easier to read (and hence much shorter).