University of Maryland
CMSC456 — Introduction to Cryptography
Professor Jonathan Katz

Lecture 9

1 More One-Way Functions/Permutations

We wrap up our diversion into number theory (for now) by presenting two more widely-used (conjectured) one-way functions.

1.1 The RSA Permutation

The first (and perhaps best-known) example is the RSA permutation. As before, let $N = pq$ be the product of two distinct primes. Recall that $\varphi(N) = |\mathbb{Z}_N^*| = (p-1)(q-1)$. Choose an arbitrary exponent e subject to the restriction that e and $\varphi(N)$ are relatively prime; i.e., $\gcd(e, \varphi(N)) = 1$. Now, there exists a $d \in \mathbb{Z}_{\varphi(N)}^*$ such that $ed = 1 \bmod \varphi(N)$ (we will not use this d in defining our function, but we will use it to prove things about the function). Define $f_{N,e} : \mathbb{Z}_N^* \rightarrow \mathbb{Z}_N^*$ by $f_{N,e}(x) = x^e \bmod N$.

Fact 1 *$f_{N,e}$ is a permutation over $\mathbb{Z}_N^*$ (note that here we do not need to restrict N, as we did for the squaring permutation).*

To see this, we prove that $f_{N,e}$ is one-to-one (and we denote $f_{N,e}$ by f for brevity). Assume $f(x) = f(y)$. Then $x^e = y^e \bmod N$. Raising both sides to the d gives: $(x^e)^d = (y^e)^d$, or $x^{ed} = y^{ed}$. Recall from a few lectures ago that we can reduce modulo the order of the group in the exponent. So, $x^{ed} = x^{ed \bmod \varphi(N)} = x^1 = x$ (and simillarly for y), giving $x = y$.

It is conjectured that the RSA permutation is one-way. (RSA was one of the first one-way functions proposed, and it hasn't been broken since then!) In mathematical notation: we assume that for all PPT algorithms A, the following is negligible:

$$\Pr[(N, e) \leftarrow \mathsf{RSAGen}(1^k); x \leftarrow \mathbb{Z}_N^*; y = x^e \bmod N : A(N, e, y) = x].$$

(RSAGen is an algorithm whose details are unimportant right now; all we know is that it generates k-bit modulus N and e relatively prime to $\varphi(N)$.)

Can we relate the hardness of inverting RSA to factoring? Well, we can say the following:

Lemma 1 *If inverting RSA is "hard" then factoring is "hard".*

Proof Assume factoring is "easy". Then we construct an efficient algorithm to invert RSA as follows: given input (N, e, y) we first factor N to find p and q. Next compute $\varphi(N) = (p-1)(q-1)$. We can now solve for $d = e^{-1} \bmod \varphi(N)$. Output $x = y^d \bmod N$ as the answer. Note that $x^e = (y^d)^e = y^{de} = y^1 = y \bmod N$, so this is indeed the correct answer. ■

On the other hand, it is not known whether the converse is true: i.e., whether factoring being hard implies that RSA is hard. In fact, there is evidence to the contrary (i.e., RSA might be easy even though factoring is hard). Thus, the RSA assumption is a (potentially) *stronger* assumption than factoring being hard.

1.2 The Discrete Logarithm Function

Before describing the function, we take a (brief) detour through more group theory. Let G be a finite group. For any element $g \in G$, define $< g > \stackrel{\text{def}}{=} \{g^0, g^1, g^2, \ldots\}$ and call this *the subgroup of G generated by g*. Note that, since G is finite, the sequence $g^0, g^1 \ldots$ will eventually start repeating (cycling). In particular, since we have $g^{|G|} = 1$, the sequence can have at most $|G|$ distinct terms in it and we can write $< g >= \{g^0, \ldots, g^{|G|-1}\}$. Of course some g will cycle before this. If $< g >$ is the entire group G we say that *g is a generator of G*. If a group G has a generator, we say that *G is cyclic*. Note that just because a group G is cyclic does not mean that every element in G is a generator.

Let's look at some examples in $\mathbb{Z}_7^*$. We have $< 1 >= \{1, 1, \ldots\} = \{1\}$. so this is pretty uninteresting! Next we have $< 2 >= \{2^0, 2^1, 2^2, 2^3, \ldots\} = \{1, 2, 4, 1, 2, 4, \ldots\} = \{1, 2, 4\}$. So 2 is not a generator of $\mathbb{Z}_7^*$. How about $< 3 >= \{3^0, 3^1, 3^2, \ldots\} = \{1, 3, 2, 6, 4, 5, 1, \ldots\} = \{1, 2, 3, 4, 5, 6\}$. So, 3 is a generator of $\mathbb{Z}_7^*$, and $\mathbb{Z}_7^*$ is cyclic.

Fix a cyclic group G and let g be a generator of G. For any element $h \in G$ we can define the *discrete logarithm of h with respect to g* as the integer k (with $0 \leq k < |G| - 1$) for which $g^k = h$ (note that because g is a generator such k always *exists* and is always *unique*). Denote this by $\log_g h = k$. This is entirely analogous to the logarithms you are familiar with from calculus, except that here we are working over a finite group and not the reals (hence the name "discrete").

We can now define our one-way function, based on the hardness of computing discrete logarithms. Let p be a prime. It is a fact that $\mathbb{Z}_p^*$ is a cyclic group (under multiplication). Let g be any generator in $\mathbb{Z}_p^*$ (we defer details to later in the semester, but note that it is known how to *efficiently* find a generator of $\mathbb{Z}_p^*$). Define $f_{p,g} : \mathbb{Z}_{p-1} \rightarrow \mathbb{Z}_p^*$ as $f_{p,g}(k) = g^k \bmod p$. It is conjectured that this is a one-way functionl in particular, no algorithm is currently known for efficiently computing discrete logarithms.

2 Back to Secure Encryption

We will not be dealing *specifically* with the one-way functions we have seen so far (factoring, squaring, RSA, discrete logarithm) when we discuss shared-key cryptography. In general, shared-key cryptosystems can be designed based on *arbitrary* one-way functions, and we will not need any special properties. However, we will use special properties of these one-way functions when we discuss public-key cryptosystems. For now, I wanted to introduce these functions so you had something concrete to think about, if this helps.

Let's recall the setting for shared-key encryption. We have Alice and Bob, who share a key sk in advance. When Alice wants to send a message m to Bob, she computes $C = \mathcal{E}_{sk}(m)$ and sends m to Bob. When Bob receives ciphertext C, he computes $m = \mathcal{D}_{sk}(C)$. In particular, for the one-time pad scheme, encryption was done by computing $C = m \oplus sk$ (where this is always bit-wise xor), and decryption was the reverse. Here, $|sk| = |m|$. We also saw that the one-time pad gives perfect secrecy, but is inefficient since we must share a key as long as the message. But, we cannot do better than the one-time pad if we want perfect secrecy.

We introduced a relaxed definition of security in an effort to get a more efficient scheme. In particular, we now want the following: that for any PPT algorithm A and for any two messages m_1, m_2 the following should be negligible:

$$\left|\Pr[sk \leftarrow \mathcal{K}(1^k); C \leftarrow \mathcal{E}_{sk}(m_1) : A(C) = 1] - \Pr[sk \leftarrow \mathcal{K}(1^k); C \leftarrow \mathcal{E}_{sk}(m_2) : A(C) = 1]\right|.$$

(I wrote $C \leftarrow \mathcal{E}_{sk}(m_1)$ because encryption might now be *randomized*.)

One way we could try to achieve this more efficiently is to follow the paradigm of the one-time pad, but instead of sharing a truly random string sk, share instead a pseudorandom string (we have not yet defined what we mean by pseudorandom, and this will require some careful thought next lecture!). In particular, imagine if we had a *pseudorandom generator* $G : \{0,1\}^{k-1} \rightarrow \{0,1\}^k$ for which the output of G "looks random" in some sense that we have yet to define. Then Alice and Bob could share a random string sk of length $k-1$, and encrypt as follows: (1) $sk' = G(sk)$, (2) $C = m \oplus sk'$ and decrypt (by reversing the steps): $m = C \oplus G(sk)$. Perhaps if the output of G looks random enough, this can be secure yet Alice and Bob share only $k-1$ bits to send a k-bit message! (So they have saved 1 bit vs. the one-time pad. Of course, this is not a big deal, but let's see if we can do this before we see how efficient we can get...)

So, our goal now is to determine whether this approach is feasible. In particular, can we come up with an appropriate definition of a pseudorandom generator such that the above scheme would be secure? And secondly, can we then construct a G which satisfies our definition? We will see in the next few classes how this can be done.