CS 7810 Graduate Cryptography October 30, 2017

Lecture 15 & 16: Trapdoor Permutations, RSA, Signatures

Lecturer: Daniel Wichs *Scribe: Willy Quach & Giorgos Zirdelis*

1 Topic Covered.

- Trapdoor Permutations.
- RSA
- Signatures from Trapdoor Permutations

2 Trapdoor Permutations (TDP).

Let us define a new basic primitive, called *Trapdoor Permutations* (TDP). Trapdoor permutations are very powerful and give us an easy way to build public-key encryption and signatures (in the RO model). On the other hand we only have very few candidates, all relying on the hardness of factoring. Moreover, TDPs don't seem to be essential in building crypto and almost all primitives that we can build from TDPs we also know how to build using other means without TDPs.

Definition 1 A *Trapdoor Permutation* (TDP) is given by 3 algorithms:

- $\mathsf{Gen}(1^n)$ outputs a pair of keys (pk, sk);
- $f_{pk} : D_{pk} \to D_{pk}$ defined over some domain D_{pk};
- $g_{sk} : D_{pk} \to D_{pk}$ defined over the same domain,

such that:

- There is an efficient sampling algorithm $x \leftarrow D_{pk}$;
- f_{pk} is a permutation computable in polynomial time (given pk);
- g_{sk} is a permutation computable in polynomial time (given sk) such that: $\forall x \in D_{pk} : g_{sk}(f_{pk}(x)) = x$.

The *security* of a TDP is expressed as:

$$\forall\, PPT\, \mathcal{A}, \Pr[\mathcal{A}(pk, f_{pk}(x)) = x \mid (pk, sk) \leftarrow \mathsf{Gen}(1^n), x \leftarrow D_{pk}] \leq \mathsf{negl}(n).$$

♢

Let us now show that Trapdoor Permutations imply Public-Key Encryption.

As a first attempt, one could think of defining (pk, sk) for the Trapdoor Permutation as the public and secret keys of the Public-Key scheme, respectively, and defining $\mathsf{Enc}_{pk}(m) := f_{pk}(m)$. This is not enough, as any CPA-secure encryption scheme cannot be deterministic.

As a second attempt, one could think of choosing a random value x, and giving $f_{pk}(x), x \oplus m$ as the ciphertext. This does not work either as it is not necessarily hard to recover *some part of* x given $f_{pk}(x)$.

However, we can use the above idea to encrypt a one-bit message, by using the *hard-core* bit $hc(x)$ instead of x as a one-time pad: $\mathsf{Enc}_{pk}(m) = (f_{pk}(x), hc(x) \oplus m)$,

Fr example, using the Goldreich-Levin hard-core bit we get:

$$\mathsf{Enc}_{pk}(m) = (f_{pk}(x), r, \langle x, r \rangle \oplus m).$$

where $x \leftarrow D_{pk}, r \leftarrow \{0,1\}^{|x|}$.

Alternatively, we can have a more efficient scheme in the Random Oracle Model where we can encrypt a long message, by defining:

$$\mathsf{Enc}_{pk}(m) = (f_{pk}(x), RO(x) \oplus m).$$

3 Cryptography modulo N = pq.

3.1 Chinese Remainder Theorem.

Define $N = pq$ for two primes p and q. The structure of $\mathbb{Z}_N$ is well understood:

Theorem 1 (Chinese Remainder Theorem (CRT)) *Let $N = pq$, where p and q are prime. Then:*

$$\mathbb{Z}_N \cong \mathbb{Z}_p \times \mathbb{Z}_q;$$
$$\mathbb{Z}_N^* \cong \mathbb{Z}_p^* \times \mathbb{Z}_q^*;$$

In other words, there exists an isomorphism $\psi : \mathbb{Z}_N \to \mathbb{Z}_p \times \mathbb{Z}_q$ defined by $a_p = a \bmod p$, and $a_q = a \bmod q$. It's easy to see that if $\psi(a) = (a_p, b_q)$, and $\psi(b) = (b_p, b_q)$, then:

- $\psi(a + b) = (a_p + b_p, a_q + b_q)$,
- $\psi(a \cdot b) = (a_p \cdot b_p, a_q \cdot b_q)$.

Furthermore, ψ is invertible and the inverse ψ^{-1} can be efficiently computed using the extended euclidean algorithm. Indeed, as p and q are primes, we can efficiently compute integers x and y such that $px + qy = 1$. This gives $px = 1 \bmod q$ and $qy = 1 \bmod p$. In other words, $\psi(px) = (0, 1)$, and $\psi(qy) = (1, 0)$, so that for all $(\alpha, \beta) \in \mathbb{Z}_p \times \mathbb{Z}_q$, $\psi^{-1}(\alpha, \beta) = \alpha qy + \beta px$.

The theorem implies in particular that $\varphi(N) = |\mathbb{Z}_N^*| = |\mathbb{Z}_p^*| \cdot |\mathbb{Z}_q^*| = (p-1) \cdot (q-1)$.

3.2 RSA Trapdoor Permutation.

Fix an integer N. Recall that by Lagrange's theorem $X^{\varphi(N)} = 1 \bmod N$.

Define $f_e(x) = x^e \bmod N = x^{e \bmod \varphi(N)} \bmod N$, which is a permutation as soon as $gcd(e, \varphi(N)) = 1$. Indeed, if that is the case, there exists $d \in \mathbb{N}$ such that $e \cdot d = 1 \bmod \varphi(N)$, which gives $f_d(f_e(x)) = x^{e \cdot d} \bmod N = x$.

This naturally gives a candidate TDP for $N = pq$ where p and q are primes:

- $\mathsf{Gen}(1^n)$: pick p, q random n-bit primes, and set $N = pq$.

  Pick e such that $gcd(e, \varphi(N)) = 1$, and compute $d = e^{-1} \bmod \varphi(N)$.

  Set $pk = (N, e)$, $sk = (N, d)$, and $D_{pk} = \mathbb{Z}_N^*$;
- Define $f_{pk}(x) = x^e \bmod N$;
- and $g_{sk}(x) = x^d \bmod N$.

We know that f_{pk} and g_{sk} are permutations that can be computed in polynomial time, and we can sample efficiently from D_{pk}.

It is not clear that this TDP is secure though; it is actually assumed, and this assumption is called the *RSA assumption*.

We often compare the RSA assumption to the hardness Factoring, defined as follows:

$$\forall\, PPT\ \mathcal{A}, \Pr[\mathcal{A}(N) = \{p, q\} \,|\, p, q \leftarrow n\text{-bit primes}, N = pq] \leq \mathsf{negl}(n).$$

It follows that solving the RSA problem is no harder than solving the Factoring problem. In other words, if Factoring is easy, then RSA is easy. Indeed, solving Factoring allows to compute $\varphi(N) = (p-1) \cdot (q-1)$, which in turn allows to compute $e^{-1} \bmod \varphi(N)$ and break the RSA assumption. The other direction is not known and it is possible that factoring is hard but RSA is easy.

This raises the natural question of building a Public-Key cryptosystem based on the hardness of Factoring alone.

3.3 Rabin's PKE

Instead of considering exponentiation by a public parameter e, we simply square the input. Define:

$$\begin{aligned} f :& \mathbb{Z}_N^* \to \mathbb{Z}_N^* \\ & x \mapsto x^2 \end{aligned}$$

This is not a permutation. In fact, the image of this function is QR_N the subgroup of quadratic residues modulo N and we will see that only 1/4th of the elements of $\mathbb{Z}_N^*$ are in QR_N. Let us spend some time understanding the structure of this function and the group QR_N.

Squaring and Quadratic Residues mod p. First, let us understand the squaring function $f(x) = x^2 \bmod p$ for a prime p and the corresponding group QR_p.

We know that for any generator g of $\mathbb{Z}_p^*$, the group QR_p consists of all of the even powers of g: $QR_p = \{1, g^2, g^4, \ldots, g^{p-3}\}$. Therefore $|QR_p| = |\mathbb{Z}_p^*|/2 = (p-1)/2$. Note that 1 has two square-roots under f namely, 1 and $g^{(p-1)/2}$. Therefore it must be the case that $g^{(p-1)/2} = -1$. In general, every element g^{2i} in QR_p has two square-roots, namely g^i and $-g^i = g^{i+(p-1)/2}$.

Suppose furthermore that $p = 3 \bmod 4$, so that $(p-1)/2$ is odd. Then, for any $g^{2i} \in QR_p$, out of the two square-roots $g^i, g^{i+(p-1)/2}$ exactly one of them has an even exponent *i.e.* exactly one preimage is in QR_p. Therefore, the squaring function f induces a permutation over QR_p.

Actually, the permutation is efficiently invertible. Indeed, write $p = 4i + 3$, and $y = x^2 \bmod p$. Then, we have $(y^{(i+1)})^2 = y^{2i+2} = x^{4i+4} = y$, so $y^{(i+1)}$ and $y^{(i+1)+(p-1)/2}$ are preimages of x, and (exactly) one of them is a quadratic residue, and therefore an inverse of $f_{|QR_p}$.

Squaring and Quadratic Residues mod N. The above generalizes directly to $\mathbb{Z}_N^*$ where $N = pq$ by the Chinese Remainder Theorem. In particular, it implies $QR_N \cong QR_p \times QR_q$ and $|QR_N| = |\mathbb{Z}_N^*|/4$.

Each ement in $y \in QR_N$ has exactly 4 square-roots. If (x_p, x_q) is one square-root of $y = (y_p, y_q) \in QR_N$ then $(\pm x_p, \pm x_q)$ are the 4 square-roots of y.

The function $f(x) = x^2$ is not injective (it is 4-to-1) when the domain is $\mathbb{Z}_N^*$. However, if we restrict the domain to QR_N and $p \equiv 3 \bmod 4$ and $q \equiv 3 \bmod 4$ then $f(x) = x^2$ is a permutation. Furthermore, f is efficiently invertible *given p and q* since we can efficiently invert modulo p,q individually.

Lastly, we can sample elements from QR_N efficiently: just sample elements from $\mathbb{Z}_N^*$, and square the samples.

We claim that $f(x) = x^2 \bmod N$ is a TDP. To prove this, we first prove the following.

Lemma 1 *Given $x, z \in \mathbb{Z}_N^*$ such that $x^2 = z^2$ but $x \neq \pm z$, one can efficiently factor N.*

Proof: Due to the expression of the 4 preimages of any $y \in QR_N$ above, we have $x = (x_p, x_q)$ and $z = (x_p, -x_q)$ or $z = (-x_p, x_q)$.

Suppose that $z = (x_p, -x_q)$. Then $x + z = (0, 2x_q)$ which is divisible by p but not by q; so that $gcd(x + z, N) = p$.

Otherwise, $x - z = (2x_p, 0)$ and the same argument applies. □

We can now prove that $f(x) = x^2$ is a one-way function over $\mathbb{Z}_N^*$. In particular, this means that given a random $y \in QR_N$ one cannot find any square-root x such that $x^2 = y \bmod N$. Note that this is a stronger statement then just saying that f is a one-way permutation over QR_N since we're showing that it's hard to find any square-root, not just the one which is itself a quadratic residue.

Theorem 2 *$f : \mathbb{Z}_N^* \to QR_N, x \mapsto x^2$ is a One-Way Function assuming Factoring is hard.*

Proof: Assume $\mathcal{A}$ breaks the one-wayness of f. Our reduction R does the following: it picks x randomly in $\mathbb{Z}_N^*$, and sends $y = x^2$ to the adversary, which outputs some z. If $z^2 = x^2$ and $z \neq \pm x$, the reduction uses the Lemma above to factor N.

The adversary finds z^2 such that $x^2 = z^2$ with non-negligible probability by definition. We saw that for any $y \in QR_N$, y has 4 different preimages from f. As a result, we have that $z \neq \pm x$ with probability $1/2$ over the randomness of x alone (as the adversary only sees $y = x^2$). So the reduction factors N with non-negligible probability. □

4 Signatures

4.1 Definition

We now define *Signatures*, which is the public-key counterpart of MACs, and present a construction from Trapdoor Permutations in the Random Oracle Model. Intuitively, it allows to *sign* a message using a secret signing key, while allowing public verification of signatures (given the associated message).

Definition 2

A *Signature* is defined by 3 algorithms:

- $\mathsf{Gen}(1^n)$: outputs a pair of keys (pk, sk),
- $\mathsf{Sign}_{sk}(m)$: outputs a signature σ,
- $\mathsf{Verify}_{\mathsf{pk}}(\mathsf{m}, \sigma)$: outputs a bit b,

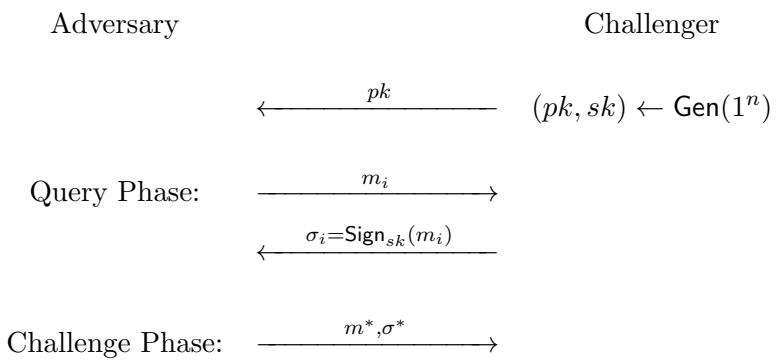
such that:

- The signature is *correct*, *i.e.*:

$$\Pr[\mathsf{Verify}_{\mathsf{pk}}(\mathsf{m}, \mathsf{Sign}_{\mathsf{sk}}(\mathsf{m})) = 1 \,|\, (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^{\mathsf{n}})] = 1.$$

- The signature is *secure*. To define security, consider the following game:

  SigGame:



and where the Adversary can request access to the Query Phase as many times as he wants.

The output of the game is 1 if $m^* \neq m_i \, \forall i$ and $\mathsf{Verify}_{\mathsf{pk}}(\mathsf{m}^*, \sigma^*) = 1$.

A Signature scheme is *secure* if:

$$\forall \, PPT \, \mathcal{A}, \Pr[\mathsf{SigGame}_{\mathcal{A}}(1^{\mathsf{n}}) = 1] = \mathsf{negl}(\mathsf{n}).$$

$\diamondsuit$

4.2 Construction from TDPs in the Random Oracle Model

As a first attempt, one might think of defining $\mathsf{Sign}_{sk} = g_{sk}(m)$, and $\mathsf{Verify}_{\mathsf{pk}}(\mathsf{m}, \sigma)$ checks that $f_{pk}(\sigma) = m$. This alone is not enough, as a forger can simply pick any σ^*, and compute $m^* = f_{pk}(\sigma)$.

The signature scheme from trapdoor permutations in the RO model is defined succinctly as follows:

- A pair of keys (pk, sk) which is the same as with trapdoor permutations just by setting $pk = \mathsf{PK}$.
- $\mathsf{Sign}(sk, m) = \mathsf{Inv}_{sk}(\mathsf{RO}(m))$
- $\mathsf{Verify}(\mathsf{VK}, \mathsf{m}, \sigma)$: $\mathsf{RO}(m) = f_{\mathsf{PK}}(\sigma)$ with $f_{\mathsf{PK}} : D_{\mathsf{PK}} \to D_{\mathsf{PK}}$ and $\mathsf{RO} : \{0,1\}^* \to D_{\mathsf{PK}}$.

To prove security, we first define a signature game, namely $\mathsf{SigGame}_{\mathcal{A}}(\mathsf{n})$ with n being the security parameter. In this game, a PPT adversary $\mathcal{A}$ is allowed to make queries to the random oracle and to the signing oracle. after that, he must come up with a new message $m^\star$ and a siganture ß for that message. A pictorial view of the game is given below:

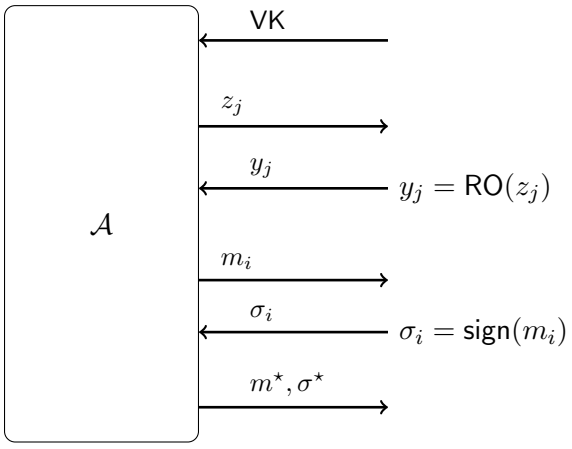


Figure 1: $\mathsf{SigGame}_{\mathcal{A}}(\mathsf{n})$.

$\mathcal{A}$ wins (or outputs 1) iff $m^\star \notin m_i$ and $\mathsf{RO}(m^\star) = f_{\mathsf{PK}}(\sigma^\star)$. We note that the adversary has access to the Random Oracle only by querying it.

Before we continue we make a couple of simplifying assumptions for which $\mathcal{A}$ maintains the same success probability and we can always convert an adversary to one that satisfies these assumptions:

1. $\mathcal{A}$ makes exactly $q = q(n)$ distinct queries to the Random Oracle
2. $\mathcal{A}$ always queries RO on m_i and $m^\star$

Our goal it to use an attacker $\mathcal{A}$ that can win $\mathsf{SigGame}_{\mathcal{A}}(\mathsf{n})$ with some non-negligible probability $\varepsilon(n)$, to create an attacker $\mathcal{B}$ that inverts the trapdoor permutation with non-negligible probability. We do this with a reduction. The usual trapdoor permutation security game is given below:

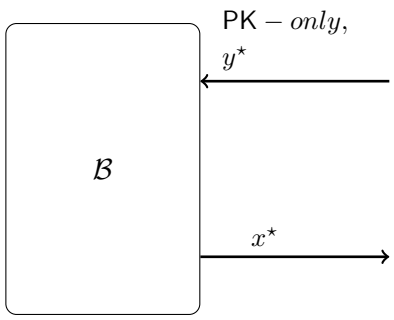


Figure 2: The trapdoor permutation game.

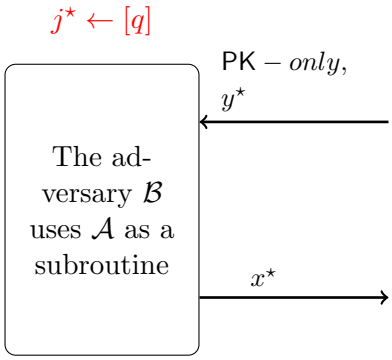
$\mathcal{B}$ wins (or outputs 1) iff $f_{\mathsf{PK}}(x^\star) = y^\star$. PK is a random public key of the trapdoor permutation and $y^\star$ is a random value from D_{PK}.

In the reduction, $\mathcal{B}$ has to give the input to $\mathcal{A}$ (or play $\mathsf{SigGame}_{\mathcal{A}}(\mathsf{n})$ with) as $\mathcal{A}$ expects it to be. The first thing that $\mathcal{B}$ does it to give $\mathcal{A}$ the key PK. After that, there are two types queries that $\mathcal{B}$ must answer to $\mathcal{A}$: random oracle and signature queries. While we will see in detail how $\mathcal{B}$ answers these queries there is a subtle point here. At some point $\mathcal{B}$ has to give $\mathcal{A}$ the image he wants to invert, i.e. $y^\star$. He has to do that when $\mathcal{A}$ queries the message $m^\star$ on the random oracle. The problem is that $\mathcal{B}$ does not know when $\mathcal{A}$ will do that. So what is the best strategy for $\mathcal{B}$ here? To make a guess! To formalize this a bit more, we define a new $\mathsf{SigGame}'_{\mathcal{A}}(\mathsf{n})$ where we pick at random a query index from 1 through q and now the probability of $\mathcal{B}$ winning the game also depends on guessing correct when $\mathcal{A}$ will query $m^\star$ on the random oracle, which essentially makes it more difficult for $\mathcal{B}$ to do so. We denote with red color the additions on the new $\mathsf{SigGame}'_{\mathcal{A}}(\mathsf{n})$ game that is illustrated below:

The probability of $\mathcal{B}$ winning $\mathsf{SigGame}'_{\mathcal{A}}(\mathsf{n})$ is,

$$\Pr[\mathsf{SigGame}'_{\mathcal{A}}(\mathsf{n}) = 1] = \Pr[\mathsf{SigGame}_{\mathcal{A}}(\mathsf{n}) = 1] \cdot \Pr[j^\star \text{ is correct} | \mathsf{SigGame}_{\mathcal{A}}(\mathsf{n}) = 1] = \frac{\varepsilon(n)}{q}.$$

This follows from the fact that $j^\star$ is random and indepentent of $\mathsf{SigGame}_{\mathcal{A}}(\mathsf{n})$ and also $\mathcal{A}$ does not know its value, hence $\Pr[j^\star \text{ is correct} | \mathsf{SigGame}_{\mathcal{A}}(\mathsf{n}) = 1] = \Pr[j^\star \text{ is correct}] = 1/q$. We assumed that the winning probability $\varepsilon(n)$ of $\mathcal{A}$ for $\mathsf{SigGame}'_{\mathcal{A}}(\mathsf{n})$ is non-negligible,



$\mathcal{B}$ wins (or outputs 1) iff $f_{\mathsf{PK}-only}(x^\star) = y^\star \wedge \ j^\star \leftarrow [q]$

Figure 3: $\mathsf{SigGame}'_{\mathcal{A}}(\mathsf{n})$.

therefore the probability of $\mathcal{B}$ winning the game is also non-negligible. Next, we give in detail how $\mathcal{B}$ works in steps (i.e. the reduction):

1. Choose $j^\star \leftarrow [q]$
2. Set $pk = \mathsf{PK}$
3. On a RO query z_j:
   (a) if $j \neq j^\star$: $\begin{cases} x_j \leftarrow D_{\mathsf{PK}} \\ \text{output } y_j = f_{\mathsf{PK}}(x_j) \end{cases}$
   (b) if $j = j^\star$: output $y_j = y^\star$
4. On sign query $m_i = z_j$:
   (a) if $j \neq j^\star$: output x_j
   (b) if $j = j^\star$: quit (or output something bogus)
5. Output $x^\star = \sigma^\star$

Some observations are in order.

- There is no RO in this game.
- Steps 3 and 4 are repeated for a total of q times.
- At step 3(a) x_j is chosen at random therefore $y_j = f_{\mathsf{PK}}(x_j)$ is also random. We store the pairs (x_j, y_j) because we need them at step 4. The same holds for 3(b), because $y^\star$ is chosen at random.

- At step 4(a) we know how to answer the sign queries, because at step 3 we chose first the x_j and the y_j. Note that we have assumed that before $\mathcal{A}$ make a sign query, he first makes a RO query on the same message (moce precisely, he makes a query to what he believes to be a RO).

By the previous analysis is that the winning probability of $\mathcal{B}$ is:

$$\Pr\left[\mathcal{B}(\mathsf{PK}, y^\star) = x^\star : (\mathsf{PK}, sk) \leftarrow \mathsf{Gen}(1^n), x^\star \leftarrow D_{\mathsf{PK}}, y^\star \leftarrow f_{\mathsf{PK}}(x^\star)\right] = \frac{\varepsilon(n)}{q}.$$