CS 6501 Week 4: Number-Theoretic Cryptography

Previously, we saw how to use the conjectured hardness of the discrete logarithm problem to construct public-key cryptography. This week, we look at another popular class of number-theoretic assumptions.

We begin by describing some facts about working in a <u>composite-order</u> group.

Let $N = pq$ be a product of two primes p, q. Then, $\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$ is the <u>additive</u> group of integers modulo N. Let $\mathbb{Z}_N^*$ be the set of integers that are invertible (under <u>multiplication</u>) modulo N.

$$x \in \mathbb{Z}_N^* \text{ if and only if } \gcd(x, N) = 1$$

Since $N = pq$ and p, q are prime, $\gcd(x, N) = 1$ unless x is a multiple of p or q:

$$|\mathbb{Z}_N^*| = N - p - q + 1 = pq - p - q + 1 = (p-1)(q-1) = \varphi(N)$$

Euler's phi function

Recall Lagrange's Theorem:

for all $x \in \mathbb{Z}_N^*$: $x^{\varphi(N)} = 1 \pmod N$

important: "ring of exponents" operate modulo $\varphi(N) = (p-1)(q-1)$

Hard problems in composite-order groups:

- <u>Factoring</u>: given $N = pq$ where p and q are sampled from a suitable distribution over primes, output p, q
- <u>Computing cube roots</u>: Sample random $x \xleftarrow{R} \mathbb{Z}_N^*$. Given $y = x^3 \pmod N$, compute $x \pmod N$.
  - This problem is <u>easy</u> in $\mathbb{Z}_p^*$ (when $3 \nmid p-1$). Namely, compute $3^{-1} \pmod{p-1}$, say using Euclid's algorithm, and then compute $y^{3^{-1}} \pmod p = (x^3)^{3^{-1}} \pmod p = x \pmod p$.
  - Why does this procedure not work in $\mathbb{Z}_N^*$. Above procedure relies on computing $3^{-1} \pmod{|\mathbb{Z}_N^*|} = 3^{-1} \pmod{\varphi(N)}$. But we do not know $\varphi(N)$ and computing $\varphi(N)$ is <u>as hard as</u> factoring N. In particular, if we know N and $\varphi(N)$, then we can write

$$\begin{cases} N = pq \\ \varphi(N) = (p-1)(q-1) \end{cases} \quad \text{[both relations hold over the integers]}$$

and solve this system of equations over the integers (and recover p, q)

Hardness of computing cube roots is the basis of the <u>RSA assumption</u>:

<u>RSA assumption</u>: Take $p, q \leftarrow \underbrace{\text{Primes}(1^\lambda)}_{\text{distribution over prime numbers.}}$, and set $N = pq$. Then, for all efficient adversaries $\mathcal{A}$,

$$\Pr[x \xleftarrow{R} \mathbb{Z}_N^*;\ y \leftarrow \mathcal{A}(N, x) : y^3 = x] = \text{negl}(\lambda)$$

more generally, can replace 3 with any e where $\gcd(e, \varphi(N)) = 1$

Hardness of RSA relies on $\varphi(N)$ being hard to compute, and thus, on hardness of factoring
(Reverse direction factoring $\overset{?}{\Longrightarrow}$ RSA is <u>not</u> known)

RSA problem gives an instantiation of more general notion called a <u>trapdoor permutation</u>:

$$F_{RSA} : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$$

$$F_{RSA}(x) := x^e \pmod N \quad \text{where } \gcd(N, e) = 1$$

Given $\varphi(N)$, we can compute $d = e^{-1} \pmod{\varphi(N)}$. Observe that given d, we can invert F_{RSA}:

$$F_{RSA}^{-1}(x) := x^d \pmod N.$$

Then, for all $x \in \mathbb{Z}_N^*$:

$$F_{RSA}^{-1}(F_{RSA}(x)) = (x^e)^d = x^{ed \pmod{\varphi(N)}} = x^1 = x \pmod N.$$

Trapdoor permutations: A trapdoor permutation (TDP) on a domain $\mathcal{X}$ consists of three algorithms:

- $\text{Setup}(1^\lambda) \to (pp, td)$: Outputs public parameters pp and a trapdoor td
- $F(pp, x) \to y$: On input the public parameters pp and input x, outputs $y \in \mathcal{X}$
- $F^{-1}(td, y) \to x$: On input the trapdoor td and input y, output $x \in \mathcal{X}$

Requirements:

- <u>Correctness</u>: for all pp output by Setup:
  - $F(pp, \cdot)$ implements a permutation on $\mathcal{X}$.
  - $F^{-1}(td, F(pp, x)) = x$ for all $x \in \mathcal{X}$.
- <u>Security</u>: $F(pp, \cdot)$ is a one-way function (to an adversary who does not see the trapdoor)

"Textbook RSA" (How <u>NOT</u> to encrypt): Consider the following candidate of a PKE scheme from trapdoor permutations:

- $\text{KeyGen}(1^\lambda)$: Sample $(pp, td) \leftarrow \text{Setup}(1^\lambda)$ for TDP and set $pk = pp$ and $sk = td$
- $\text{Encrypt}(pk, m)$: Output $F(pk, m)$
- $\text{Decrypt}(sk, ct)$: Output $F^{-1}(sk, ct)$

Correctness follows from correctness of TDP.

How about security? <u>NO</u>.

1. Security of TDP says that inverting <u>random</u> element should be difficult
   - ↳ Does not apply if messages chosen adversarially (e.g., semantic security definition)
   - ↳ Does not say anything about hiding preimage (e.g., $F(pp, x)$ can leak information about x so long as leakage is not sufficient to fully recover x — this is a weaker property than full indistinguishability)
2. This scheme is <u>deterministic</u>: cannot be semantically secure!

<u>NEVER</u> use textbook RSA! See HW2 also for additional attacks on textbook RSA and simple variants.

How to encrypt using TDP? Need to leverage "hard-to-invert" property to obtain something indistinguishable from uniform.

<u>Idea</u>: Apply a random oracle to derive a pad to blind a message.

Let $\mathcal{X}$ be domain/range of TDP, $\{0,1\}^n$ be the message space and $H : \mathcal{X} \to \{0,1\}^n$ be a hash function (modeled as random oracle).

$\text{KeyGen}(1^\lambda)$: Sample $(pp, td) \leftarrow \text{Setup}(1^\lambda)$ and output $pk = pp$ and $sk = td$.

$\text{Encrypt}(pk, m)$: Sample $x \xleftarrow{R} \mathcal{X}$. Output the ciphertext $ct = (F(pk, x), m \oplus H(x))$.

$\text{Decrypt}(sk, ct)$: Output $ct_1 \oplus H(F^{-1}(sk, ct_0))$.

Important: The TDP is only applied to <u>random</u> elements, <u>not</u> to the <u>message</u> (which is adversarially chosen)

<u>Correctness</u>: Follows by correctness of TDP. Namely, if $ct \leftarrow \text{Encrypt}(pk, m)$, then $ct = (F(pp, x), m \oplus H(x))$ and so

$$\begin{aligned} \text{Decrypt}(sk, ct) &= m \oplus H(x) \oplus H(F^{-1}(td, F(pp, x))) \\ &= m \oplus H(x) \oplus H(x) \\ &= m \end{aligned}$$

<u>Security</u>. Informally, given a ciphertext, message m is information-theoretically hidden from the adversary unless it makes a query to the random orale at input x (given only $F(pp, x)$). Since x is chosen uniformly, such an adversary breaks security of TDF.

Public-key encryption is the analog of symmetric encryption in the public-key setting. What about authentication? Can we define a "public-key" MAC?

↳ Concept of a <u>digital signature</u>. Holder of <u>secret signing key</u> can generate signatures, but everyone can <u>publicly</u> verify signatures.

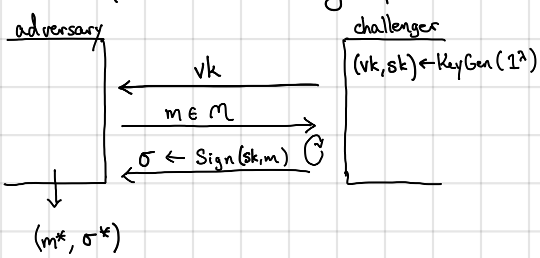
↳ Applications: software updates / distribution (patch is certified by developer and OS verifies before installing)
authenticated key exchange (server includes a signed certificate as proof of its identity during key agreement)

Digital signature scheme: Consists of three algorithms:

- $\text{KeyGen}(1^\lambda) \to (vk, sk)$: Outputs a verification key vk and a signing key sk
- $\text{Sign}(sk, m) \to \sigma$: Takes the signing key sk and a message m and outputs a signature σ
- $\text{Verify}(vk, m, \sigma) \to 0/1$: Takes the verification key vk, a message m, and a signature σ, and outputs a bit $0/1$

Two requirements:

- <u>Correctness</u>: For all messages $m \in \mathcal{M}$, $(vk, sk) \leftarrow \text{KeyGen}(1^\lambda)$, then
$$\Pr[\text{Verify}(vk, m, \text{Sign}(sk, m)) = 1] = 1.$$
[Honestly-generated signatures <u>always</u> verify]
- <u>Unforgeability</u>: Very similar to MAC security. For all efficient adversaries $\mathcal{A}$, $\text{SigAdv}[\mathcal{A}] = \Pr[W = 1] = \text{negl}(\lambda)$, where W is the output of the following experiment:



Let $m_1, \ldots, m_Q$ be the signing queries the adversary submits to the challenger Then, $W = 1$ if and only if:
$$\text{Verify}(vk, m^*, \sigma^*) = 1 \quad \text{and} \quad m^* \notin \{m_1, \ldots, m_Q\}$$
Adversary cannot produce a valid signature on a <u>new</u> message.

Digital signatures from TDPs (in the RO model).

Let $\mathcal{M}$ be the message space and $\mathcal{X}$ be the domain/range of a TDP. Let $H: \mathcal{M} \to \mathcal{X}$ be a hash function (modeled as RO).

- $\text{KeyGen}(1^\lambda)$: Sample $(pp, td) \leftarrow \text{Setup}(1^\lambda)$ for the TDP. Output $vk = pp$ and $sk = td$.
- $\text{Sign}(sk, m)$: On input the signing key sk and a message m, output $\sigma \leftarrow F^{-1}(sk, H(m))$.
- $\text{Verify}(vk, m, \sigma)$: On input the verification key vk, the message m, and the signature σ, output 1 (i.e., valid signature) if $H(m) = F(pp, \sigma)$ and 0 otherwise.

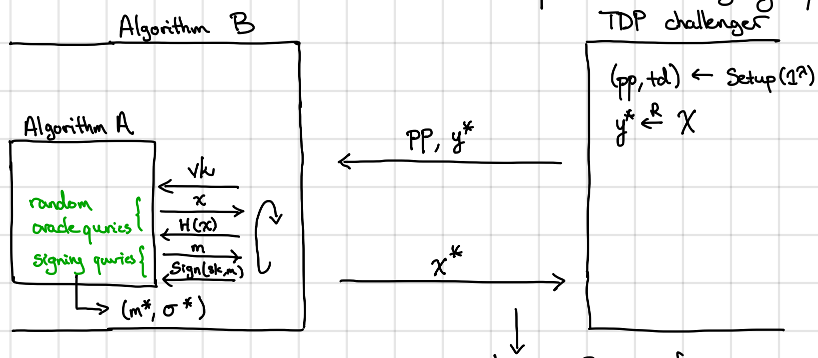
<u>Correctness</u>. Follows by correctness of TDP. In particular, if $\sigma \leftarrow \text{Sign}(sk, m)$, then $F(pp, \sigma) = F(pp, F^{-1}(td, H(m))) = H(m)$.

<u>Security</u>. Intuitively, to forge a signature on a message m, adversary has to invert TDP on $H(m)$ and since $H(m)$ is uniformly random, this is difficulty by security of the TDP. Actual security proof will rely on "programming the random oracle."

Theorem. If $(\text{Setup}, F, F^{-1})$ is a secure TDP and H is modeled as a random oracle, then $(\text{KeyGen}, \text{Sign}, \text{Verify})$ is a secure signature scheme.

Proof (Sketch). We show that if there exists a signature adversary A, then there is an adversary B that inverts the TDP. Algorithm B needs to simulate both random oracle queries and signing queries for A.

We need to describe has to simulate these queries



algorithm B wins if
$F(pp, x^*) = y^*$

Will make some simplifying assumptions <u>without loss of generality</u>:
- Algorithm A makes RO query on message m prior to a signing query on m
- Algorithm A makes RO query on message m^* at some point in the game

} Any adversary that does not conform to this schema can be converted into one that is conforming

Algorithm B works as follows:

1. Let Q be a bound on the number of random oracle queries the adversary makes. Algorithm B chooses a random index $i^* \leftarrow [Q]$ (this is where A will invert the TDP / produce its forgery).
2. Simulating the verification key vk: set $vk = pp$.
3. When A makes its i^{th} RO query (on message m):
   - if $i = i^*$: respond with the TDP challenge y
   - otherwise: sample random $x_m \xleftarrow{R} X$ and reply with $y_m = F(pp, x)$
4. When A makes a signing query for message m:
   - by assumption, A previously queried the random oracle on m
   - if $H(m)$ was not the i^{th} RO query, B can reply with x_m (otherwise, B aborts the simulation)

     since $F^{-1}(td, y_m) = x_m$ ← $H(m)$
5. If m^* was RO query i^*, algorithm B outputs σ^* as its response

two observations:
- value of random oracle is uniformly random (correctly simulated)
- simulate $H(m)$ such that B knows the preimage of $H(m)$ (so signatures can be simulated)

Recall that A <u>cannot</u> make signing query on m^* (the challenge message)

Simulator chooses values of RO so that it can generate signatures (since it cannot invert TDP itself)

Analysis:
- Suppose we guess correctly (A queries RO on m^* in query i^*). Then, all queries answered <u>perfectly</u> and Algorithm A outputs σ^* which is a valid signature on m^* with non-negligible probability. This means that $F(pp, \sigma^*) = H(m^*) = y^*$, in which case B wins the TDP security game
- If we guess wrong, then B fails.
- $\text{SigAdv}[B] = \frac{1}{Q}\,\text{TDPAdv}[A]$

  ↳ Q is number of queries A makes, and B guesses correctly with probability $\frac{1}{Q}$

Recap:
- TDPs are useful building blocks for constructing public-key primitives (both PKE and digital signatures)
- TDPs can be built from the RSA assumption (using composite-order groups and relies implicitly on the hardness of factoring)
- RSA/factoring gives the <u>only</u> known instantiation of TDPs
  - ↳ <u>Open Question</u>: Constructions from other assumptions?