CSE107: Intro to Modern Cryptography

https://cseweb.ucsd.edu/classes/sp22/cse107-a/

Emmanuel Thomé

May 5, 2022

Lecture 10b

RSA

Plan

RSA Math

Recall that $\varphi(N) = |\mathbb{Z}_N^*|$.

Claim: Suppose $e, d \in \mathbb{Z}^*_{\varphi(N)}$ satisfy $ed \bmod \varphi(N) = 1$. Then for any $x \in \mathbb{Z}_N^*$ we have

$$(x^e)^d \bmod N = x\ .$$

Proof:

$$\begin{aligned}(x^e)^d \bmod N &= x^{ed \bmod \varphi(N)} \bmod N \\ &= x^1 \bmod N = x\end{aligned}$$

The RSA function

A modulus N and encryption exponent $e \in \mathbb{Z}^*_{\varphi(N)}$ define the RSA function $f : \mathbb{Z}^*_N \to \mathbb{Z}^*_N$ via:

$$f(x) = x^e \bmod N$$

for all $x \in \mathbb{Z}^*_N$.

A value $d \in Z^*_{\varphi(N)}$ satisfying $ed \bmod \varphi(N) = 1$ is called a decryption exponent.

Claim: The RSA function $f : \mathbb{Z}^*_N \to \mathbb{Z}^*_N$ is a permutation with inverse $f^{-1} : \mathbb{Z}^*_N \to \mathbb{Z}^*_N$ given by

$$f^{-1}(y) = y^d \mod N$$

Proof: For all $x \in \mathbb{Z}^*_N$, the prior claim says that we have

$$f^{-1}(f(x)) = (x^e)^d \bmod N = x \ .$$

Example

Let $N = 15$. So

$$\begin{aligned} \mathbb{Z}_N^* &= \{1, 2, 4, 7, 8, 11, 13, 14\} \\ \varphi(N) &= 8 \\ \mathbb{Z}_{\varphi(N)}^* &= \{1, 3, 5, 7\} \end{aligned}$$

Example

Let $N = 15$. So

$$\begin{aligned} \mathbb{Z}_N^* &= \{1,2,4,7,8,11,13,14\} \\ \varphi(N) &= 8 \\ \mathbb{Z}_{\varphi(N)}^* &= \{1,3,5,7\} \end{aligned}$$

Let $e = 3$ and $d = 3$. Then

$$ed \equiv 9 \equiv 1 \pmod 8$$

Let

$$\begin{aligned} f(x) &= x^3 \bmod 15 \\ g(y) &= y^3 \bmod 15 \end{aligned}$$

x	$f(x)$	$g(f(x))$
1	1	1
2	8	2
4	4	4
7	13	7
8	2	8
11	11	11
13	7	13
14	14	14

Trapdoor permutation

RSA is a trapdoor, one-way permutation:

- Easy to invert given trapdoor d
- Hard to invert given only N, e

The second is true, to best of our current knowledge, for appropriately-chosen parameters N, e, d.

The choice of parameters is done by an algorithm called an RSA generator.

RSA generators

An RSA generator with security parameter k is an algorithm $\mathcal{K}_{rsa}$ that returns N, p, q, e, d satisfying

- p, q are distinct odd primes
- $N = pq$, and is called the (RSA) modulus
- $|N| = k$, meaning $2^{k-1} \leq N \leq 2^k$
- $e \in \mathbb{Z}^*_{\varphi(N)}$ is called the encryption exponent
- $d \in \mathbb{Z}^*_{\varphi(N)}$ is called the decryption exponent
- $ed \bmod \varphi(N) = 1$

A formula for Phi

Fact: Suppose $N = pq$ for distinct primes p and q. Then

$$\varphi(N) = (p-1)(q-1)\ .$$

Example: Let $N = 15 = 3 \cdot 5$. Then the Fact says that

$$\varphi(15) = (3-1)(5-1) = 8.$$

As a check, $\mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$ indeed has size 8.

A more general formula for Phi

Fact: Suppose $N \geq 1$ factors as

$$N = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_n^{\alpha_n}$$

where $p_1 < p_2 < \ldots < p_n$ are primes and $\alpha_1, \ldots, \alpha_n \geq 1$ are integers. Then

$$\begin{aligned}\varphi(N) &= p_1^{\alpha_1 - 1}(p_1 - 1) \cdot p_2^{\alpha_2 - 1}(p_2 - 1) \cdot \ldots \cdot p_n^{\alpha_n - 1}(p_n - 1) \\ &= N \times \left(1 - \frac{1}{p_1}\right) \times \left(1 - \frac{1}{p_2}\right) \times \cdots \times \left(1 - \frac{1}{p_n}\right) .\end{aligned}$$

Note prior Fact is a special case of the above.

Example: Let $N = 45 = 3^2 \cdot 5^1$. Then the Fact says that

$$\varphi(45) = 3^1(3-1) \cdot 5^0(5-1) = 24$$

Recall

Given $\varphi(N)$ and $e \in \mathbb{Z}^*_{\varphi(N)}$, we can compute $d \in \mathbb{Z}^*_{\varphi(N)}$ satisfying $ed \bmod \varphi(N) = 1$ via

$$d \leftarrow \text{MOD-INV}(e, \varphi(N)).$$

We have algorithms to efficiently test whether a number is prime, and we know that a random number has a pretty good chance of being a prime.

We use these facts to build RSA generators.

Building RSA generators

Say we wish to have $e = 3$. (We will see that the smaller is e, the more efficient is encryption.)

The generator $\mathcal{K}_{\text{rsa}}^{e=3}$ with (even) security parameter k is as follows:

```
Alg $\mathcal{K}_{\text{rsa}}^{e}(k)$   // $e$ is our public-key exponent parameter
repeat
  $p \xleftarrow{\$} \{2^{k/2-1}, \ldots, 2^{k/2} - 1\}$;
  $q \xleftarrow{\$} \{2^{k/2-1}, \ldots, 2^{k/2} - 1\}$;
  $N \leftarrow pq$; $M \leftarrow (p-1)(q-1)$
until $N \geq 2^{k-1}$ and $p, q$ are prime and $gcd(e, M) = 1$
$d \leftarrow$ MOD-INV$(e, M)$
return $N, p, q, e, d$
```

Plan

Plan

One-wayness of RSA

The following should be hard:

Given: N, e, y where $y = f(x) = x^e \mod N$

Find: x

Formalism picks x at random and generates N, e via an RSA generator.

One-wayness of RSA, formally

Let $\mathcal{K}_{\mathrm{rsa}}$ be a RSA generator.

Game $\mathrm{OW}_{\mathcal{K}_{\mathrm{rsa}}}$

```
procedure Initialize
$(N, p, q, e, d) \xleftarrow{\$} \mathcal{K}_{\mathrm{rsa}}$
$x \xleftarrow{\$} \mathbb{Z}_N^*$; $y \leftarrow x^e \mod N$
return $N, e, y$
```

```
procedure Finalize($x'$)
return $(x = x')$
```

Definition (ow-advantage $\mathbf{Adv}^{\mathrm{ow}}$)

The ow-advantage of an adversary A is

$$\mathbf{Adv}^{\mathrm{ow}}_{\mathcal{K}_{\mathrm{rsa}}}(A) = \Pr\left[\mathrm{OW}^{A}_{\mathcal{K}_{\mathrm{rsa}}} \Rightarrow \mathsf{true}\right]$$

Inverting RSA

Inverting RSA : given N, e, y find x such that $x^e \bmod N = y$

Inverting RSA

$$\begin{tikzcd} \text{Inverting RSA} \arrow[r, phantom, ":"] & \text{given } N, e, y \text{ find } x \text{ such that } x^e \bmod N = y \\ & \text{because } x = y^d \bmod N \\ \text{Know } d \arrow[uu, "\text{EASY}"'] & \end{tikzcd}$$

Inverting RSA

$$\begin{tikzcd}
\text{Inverting RSA} & \text{: given } N, e, y \text{ find } x \text{ such that } x^e \bmod N = y \\
\text{Know } d \arrow[u, "\text{EASY}"'] & \text{because } x = y^d \bmod N \\
\text{Know } \varphi(N) \arrow[u, "\text{EASY}"'] & \text{because } d = \text{MOD-INV}(e, \varphi(N))
\end{tikzcd}$$

Inverting RSA

Inverting RSA : given N, e, y find x such that $x^e \bmod N = y$

$$\begin{tikzcd}
\text{Inverting RSA} & \\
\text{Know } d \arrow[u, "\text{EASY}"'] & \text{because } x = y^d \bmod N \\
\text{Know } \varphi(N) \arrow[u, "\text{EASY}"'] & \text{because } d = \text{MOD-INV}(e, \varphi(N)) \\
\text{Know } p, q \arrow[u, "\text{EASY}"'] & \text{because } \varphi(N) = (p-1)(q-1)
\end{tikzcd}$$

Inverting RSA

$$\begin{tikzcd}
\text{Inverting RSA} \quad : \quad \text{given } N, e, y \text{ find } x \text{ such that } x^e \bmod N = y \\
\text{Know } d \arrow[u, "\text{EASY} \qquad \text{because } x = y^d \bmod N"'] \\
\text{Know } \varphi(N) \arrow[u, "\text{EASY} \qquad \text{because } d = \text{MOD-INV}(e, \varphi(N))"'] \\
\text{Know } p, q \arrow[u, "\text{EASY} \qquad \text{because } \varphi(N) = (p-1)(q-1)"'] \\
\text{Know } N \arrow[u, "\textbf{?}"']
\end{tikzcd}$$

Plan

Factoring Problem

Given: N where $N = pq$ and p, q are prime

Find: p, q

If we can factor we can invert RSA. We do not know whether the converse is true, meaning whether or not one can invert RSA without factoring.

A factoring algorithm

```
Alg FACTOR($N$)    // $N = pq$ where $p, q$ are primes
for $i = 2, \ldots, \lceil \sqrt{N} \rceil$ do
  if $N \bmod i = 0$ then
    $p \leftarrow i$ ; $q \leftarrow N/i$ ; return $p, q$
```

This algorithm works but takes time

$$\mathcal{O}(\sqrt{N}) = \mathcal{O}(e^{0.5 \ln N})$$

which is prohibitive.

Factoring algorithms

Algorithm	**Time taken to factor N**
Naive	$O(e^{0.5 \ln N})$
Quadratic Sieve (QS)	$O(e^{c(\ln N)^{1/2}(\ln \ln N)^{1/2}})$
Number Field Sieve (NFS)	$O(e^{1.92(\ln N)^{1/3}(\ln \ln N)^{2/3}})$

Factoring records

bit-length of number	When factored	Algorithm used
400	1993	QS
428	1994	QS
431	1996	NFS
465	1999	NFS
512	1999	NFS
576	2003	NFS
768	2009	NFS
795	2019	NFS
829	2020	NFS

Moduli sizes

We estimate that a 1024-bit RSA modulus provides 80 bits of security, meaning factoring it takes 2^{80} time.

Factorization of a 1024-bit modulus hasn't been done yet in public, but is within reach of large organizations. Longer moduli, like 2048 bits, have been recommended since around 2010.

Plan

Choices of encryption exponent

Common choices are $e = 3$, $e = 17$ and $e = 65,537$. Why these?

e	bin(e)
3	11
17	10001
65,537	10000000000000001

Recall that the modular exponentiation algorithm computing $x \mapsto x^e \bmod N$ uses $c(b)$ modular multiplications per bit $b \in \{0, 1\}$ in the binary expansion bin(e), where $c(0) = 1$ and $c(1) = 2$. So the fewer the number of 1s in bin(e), the faster is the operation.

Low-exponent and other attacks

Further attacks on RSA include

- Coppersmith's attack
- Franklin-Reiter attack
- Håstad attack

These work for small encryption exponents but do not violate OW-security.

If RSA-based public-key encryption and digital signature schemes use RSA appropriately, these attacks do not threaten them, even if the encryption exponent is small.

Accordingly, in designing RSA-based public-key encryption and digital signature schemes, we seek proofs of security based (only) on the OW-security of RSA.

Plan

RSA Video

http://www.youtube.com/watch?v=wXB-V_Keiu8

RSA: what to remember

The RSA function $f(x) = x^e \mod N$ is a trapdoor one way permutation:

- Easy forward: given N, e, x it is easy to compute $f(x)$
- Easy back with trapdoor: Given N, d and $y = f(x)$ it is easy to compute $x = f^{-1}(y) = y^d \mod N$
- Hard back without trapdoor: Given N, e and $y = f(x)$ it is hard to compute $x = f^{-1}(y)$

The quantum threat

On a quantum computer, Shor's algorithm can compute discrete logarithms and factor in polynomial time.

Quantum computer capable of running Shor's algorithm don't exist, but efforts to build quantum computers that scale are underway.

RSA and DH can be replaced by elliptic curve cryptography, which has much smaller keys because the DLog problem is a lot harder on elliptic curves. However, elliptic curves are also threatened by quantum computers.

Efforts are underway to standardize public-key cryptography based on computational problems like finding short vectors in lattices for which there are currently no known efficient quantum algorithms.

CSE107: Intro to Modern Cryptography

https://cseweb.ucsd.edu/classes/sp22/cse107-a/

Emmanuel Thomé

May 5, 2022

Lecture 11

Public-Key Encryption Schemes

Security notions for PKE schemes

Two settings

Symmetric encryption:

- Before Alice and Bob can communicate securely, they need to have a common secret key K_{AB}.
- If Alice wishes to also communicate with Charlie then she and Charlie must also have another common secret key K_{AC}.
- If Alice generates K_{AB}, K_{AC}, they must be communicated to her partners over private and authenticated channels.

Asymmetric (public-key) encryption:

- Alice has a secret decryption key dk that is shared with nobody, and an associated public encryption key ek that is known to everybody.
- Anyone (Bob, Charlie, ...) can use Alice's encryption key ek to send her an encrypted message which only she can decrypt.

Syntax of a PKE scheme

A public-key (or asymmetric) encryption scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of three algorithms that operate as follows:

- $(ek, dk) \xleftarrow{\$} \mathcal{K}$ — generate an encryption key ek and matching decryption key dk
- $C \xleftarrow{\$} \mathcal{E}_{ek}(M)$ — encrypt message M under encryption key ek to get a ciphertext C. Algorithm $\mathcal{E}$ may be randomized.
- $M' \leftarrow \mathcal{D}_{dk}(C)$ — decrypt ciphertext C under decryption key dk to get an output $M' \in \{0,1\}^* \cup \{\perp\}$.

Correct decryption requirement

Let $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an asymmetric encryption scheme. The correct decryption requirement is that

$$\Pr[\mathcal{D}_{dk}(\mathcal{E}_{ek}(M)) = M] = 1$$

for all (ek, dk) that may be output by $\mathcal{K}$ and all messages M in the *message space* of $\mathcal{AE}$. The probability is over the random choices of $\mathcal{E}$.

This simply says that decryption correctly reverses encryption to recover the message that was encrypted. When we specify schemes, we indicate what is the message space.

How it works

Step 1: Key generation

Alice locally computes $(ek, dk) \xleftarrow{\$} \mathcal{K}$ and stores dk.

Step 2: Alice enables any prospective sender to get ek.

Step 3: The sender encrypts a message M under ek and sends the ciphertext C to Alice.

Step 4: Alice decrypts C under dk to recover M.

Verifying the authenticity of public keys

We don't require privacy of ek but we do require authenticity: the sender should be assured ek is really Alice's key and not someone else's.

This is solved in a variety of imperfect ways in practice. One could:

- Use certificates as we will see later. (TLS)
- Verify a fingerprint/hash of the public key through an out of band channel. (Signal)
- Use existing trusted infrastructure, like social media accounts. (Keybase)
- Build a social network of digital signatures. (PGP)
- Trust on first use and verify fingerprint/hash in the future. (SSH)

Plan

Security notions for PKE schemes

Security of PKE Schemes

We formalize two goals:

- **IND-CPA**: Indistinguishability under chosen-plaintext attack. Just like for symmetric encryption, except that adversary needs to be given the encryption key.
- **IND-CCA**: Indistinguishability under chosen-ciphertext attack. A stronger goal in which the adversary also has (limited) access to a decryption oracle.

PKE IND-CPA security games

Let $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a PKE scheme.

Game $\mathrm{Left}_{\mathcal{AE}}$

```
procedure Initialize
$(ek, dk) \xleftarrow{\$} \mathcal{K}$
return $ek$
procedure LR($M_0, M_1$)
return $\mathcal{E}_{ek}(M_0)$
```

Game $\mathrm{Right}_{\mathcal{AE}}$

```
procedure Initialize
$(ek, dk) \xleftarrow{\$} \mathcal{K}$
return $ek$
procedure LR($M_0, M_1$)
return $\mathcal{E}_{ek}(M_1)$
```

Definition (ind-cpa advantage $\mathbf{Adv}^{\text{ind-cpa}}$, public-key version)

The (ind-cpa) advantage of an adversary A is

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{AE}}(A) = \Pr\left[\mathrm{Right}^{A}_{\mathcal{AE}} \Rightarrow 1\right] - \Pr\left[\mathrm{Left}^{A}_{\mathcal{AE}} \Rightarrow 1\right]$$

PKE IND-CPA: Explanations

The "return ek" statement in **Initialize** means the adversary A gets the encryption key ek as input. It does not get dk.

It can call **LR** with any equal-length messages M_0, M_1 of its choice to get back an encryption $C \xleftarrow{\$} \mathcal{E}_{ek}(M_b)$ of M_b under ek, where $b = 0$ in game $\mathrm{Left}_{\mathcal{AE}}$ and $b = 1$ in game $\mathrm{Right}_{\mathcal{AE}}$. Notation indicates encryption algorithm may be randomized.

A is not allowed to call **LR** with messages M_0, M_1 of unequal length. Any such A is considered invalid and its advantage is undefined or 0.

It outputs a bit, and wins if this bit equals b.

Questions:

- Since ek is public, why can't the adversary run **LR** by itself?

PKE IND-CPA: Explanations

The "return ek" statement in **Initialize** means the adversary A gets the encryption key ek as input. It does not get dk.

It can call **LR** with any equal-length messages M_0, M_1 of its choice to get back an encryption $C \xleftarrow{\$} \mathcal{E}_{ek}(M_b)$ of M_b under ek, where $b = 0$ in game $\mathrm{Left}_{\mathcal{AE}}$ and $b = 1$ in game $\mathrm{Right}_{\mathcal{AE}}$. Notation indicates encryption algorithm may be randomized.

A is not allowed to call **LR** with messages M_0, M_1 of unequal length. Any such A is considered invalid and its advantage is undefined or 0.

It outputs a bit, and wins if this bit equals b.

Questions:

- Since ek is public, why can't the adversary run **LR** by itself? Because the bit b is secret!
- Can we have IND-CPA security if $\mathcal{E}$ is deterministic?

PKE IND-CPA: Explanations

The "return ek" statement in **Initialize** means the adversary A gets the encryption key ek as input. It does not get dk.

It can call **LR** with any equal-length messages M_0, M_1 of its choice to get back an encryption $C \stackrel{\$}{\leftarrow} \mathcal{E}_{ek}(M_b)$ of M_b under ek, where $b = 0$ in game $\mathrm{Left}_{\mathcal{AE}}$ and $b = 1$ in game $\mathrm{Right}_{\mathcal{AE}}$. Notation indicates encryption algorithm may be randomized.

A is not allowed to call **LR** with messages M_0, M_1 of unequal length. Any such A is considered invalid and its advantage is undefined or 0.

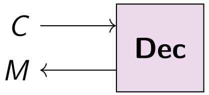
It outputs a bit, and wins if this bit equals b.

Questions:

- Since ek is public, why can't the adversary run **LR** by itself? Because the bit b is secret!
- Can we have IND-CPA security if $\mathcal{E}$ is deterministic? NO!

Chosen-ciphertext attacks

In addition to the **LR** oracle, the adversary now has access to a second oracle which can decrypt messages.



Dec knows the decryption key dk and returns $M \leftarrow \mathcal{D}_{dk}(C)$.

Adversary's goal is to learn partial information about un-decrypted messages from their ciphertexts.

PKE IND-CCA security games

Let $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a PKE scheme.

Game $\text{Left}_{\mathcal{AE}}$

```
procedure Initialize
$(ek, dk) \xleftarrow{\$} \mathcal{K}$ ; $S \leftarrow \emptyset$ ; return $ek$
procedure LR($M_0, M_1$)
$C \xleftarrow{\$} \mathcal{E}_{ek}(M_0)$ ; $S \leftarrow S \cup \{C\}$
return $C$
procedure Dec($C$)
if $C \in S$ then return $\perp$
else $M = \mathcal{D}_{dk}(C)$ ; return $M$
```

Game $\text{Right}_{\mathcal{AE}}$

```
procedure Initialize
$(ek, dk) \xleftarrow{\$} \mathcal{K}$ ; $S \leftarrow \emptyset$ ; return $ek$
procedure LR($M_0, M_1$)
$C \xleftarrow{\$} \mathcal{E}_{ek}(M_1)$ ; $S \leftarrow S \cup \{C\}$
return $C$
procedure Dec($C$)
if $C \in S$ then return $\perp$
else $M = \mathcal{D}_{dk}(C)$ ; return $M$
```

Definition (ind-cca advantage $\mathbf{Adv}^{\text{ind-cca}}$)

The ind-cca advantage of an adversary A is

$$\mathbf{Adv}^{\text{ind-cca}}_{\mathcal{AE}}(A) = \Pr\left[\text{Right}^{A}_{\mathcal{AE}} \Rightarrow 1\right] - \Pr\left[\text{Left}^{A}_{\mathcal{AE}} \Rightarrow 1\right]$$

PKE IND-CCA: Explanations

Just like IND-CPA, except that the adversary now also has access to a decryption oracle **Dec**.

It can call **Dec** with any ciphertext C of its choice.

To prevent trivial attacks, **Dec**(C) returns $\perp$ if C was in the set S, meaning had been previously returned by the **LR** oracle.

Remarks

IND-CPA is recovered as IND-CCA with adversaries restricted to make zero **Dec** queries. This shows that IND-CCA implies IND-CPA: any scheme satisfying the former also satisfies the latter.

The converse is false. As an exercise, assume you are given an IND-CPA PKE scheme, and construct another PKE scheme that (1) is IND-CPA secure, but (2) is not IND-CCA secure.

Modern applications and usage of PKE call for IND-CCA, and it is now the canonical and accepted goal for PKE.

IND-CCA can be defined also for symmetric encryption schemes, and is implied by IND-CPA+INT-CTXT.