Cryptography 2017
Lecture 5

- Textbook RSA - The Factoring Problem
- Prime numbers - Primality test
What have seen? What are we discussing today? What is coming later?

Lecture 4
- Attacks against Block Cipher Modes
- Intro to Public Key Cryptography
- Public Key Infrastructures - Certification Authorities
- Textbook RSA

Lecture 5
- PKE - Digital Signatures
- Recap on Textbook RSA
- The Factoring Problem
- Prime numbers - Primality tests

Lecture 6
- Introduction to Number Theory
Reminder: Intro to Public Key Encryption

**Alice**

- Plaintext
- **Encryption**
- **Ciphertext**

**Bob**

- **Bob’s Public Key**
- **Secret Key**
- **Bob’s Private Key**
- **Decryption**

- **Ciphertext**
- **See you today at 5pm**

**Intuition:** Anyone can encrypt a message for Bob (using Bob’s public key)

- Only Bob with his private (secret) key, can decrypt correctly!
- The public key is **known** to everyone!
- The private (secret) key must remain secret!
- The **public key** depends on the **secret** key!
Preview: Digital Signatures in PKC

**Intuition:** Only Alice with her **secret key** can **sign** any message. Everyone can **verify** if a message is signed by Alice, using her **public key**. The **encryption** algorithm is used for **signing** but with the **secret key**! The **decryption** algorithm is used to **verify** a signature but with the **public key**!

(You will sign Assignment 1 before submitting it!)
Data confidentiality vs. data integrity

Encryption

**Main goal:** Data confidentiality
Prevent the adversary from finding out info about the content of the plaintext messages.

**How does it work?**

<table>
<thead>
<tr>
<th>KeyGen</th>
<th>Encrypt</th>
<th>Decrypt</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyGen</td>
<td>sk</td>
<td>(sk, m) → c (sk, c) → m</td>
</tr>
<tr>
<td>Encrypt</td>
<td>(pk, sk)</td>
<td>(pk, m) → c (sk, c) → m</td>
</tr>
</tbody>
</table>

Digital signatures & MACs

**Main goal:** Data integrity & authentication
Prevent unauthorised modification of messages

**Non-repudiation:** no-one can deny (everyone can verify) that someone has signed a message

**How does it work?**

<table>
<thead>
<tr>
<th>KeyGen</th>
<th>Sign</th>
<th>Verify</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyGen</td>
<td>sk</td>
<td>(sk, m) → tag (sk, m, tag) → yes/no</td>
</tr>
<tr>
<td>Sign</td>
<td>(pk, sk)</td>
<td>(sk, m) → σ (pk, m, σ) → yes/no</td>
</tr>
</tbody>
</table>

Reminder: MACs stands for Message Authentication Codes
More info about MACs and Signatures in lec10 and lec11.
The founders of RSA

- RSA was first published in 1977
- Rivest, Shamir and Adleman recipients of Turing Award in 2002!!
- RSA used extensively in internet security (e.g., TLS/SSL)
Textbook RSA encryption Scheme (Rivest-Shamir-Adleman)

isko useful for the exercises/exams!

Textbook RSA

KeyGen($\lambda$) $\rightarrow$ (pk, sk)

1. generate two distinct $\lambda$-bit primes $p$ and $q$, compute $N = pq$ and $\Phi(N)$.

2. choose an integer $e \leftarrow \mathbb{Z}_{\Phi(N)}$ such that $\text{GCD}(e, \Phi(N)) = 1$ and compute its (modular) inverse $d = e^{-1} \mod \Phi(N)$.

3. set: $\textbf{pk} = (N, e)$ and $\textbf{sk} = (N, d)$

$\text{Enc(pk, m)} \rightarrow c : \text{compute } c = m^e \mod N$

$\text{Dec(sk, m)} \rightarrow m : \text{compute } m = c^d \mod N$

$\Phi$ denotes Euler’s phi (or Totient) function (more info about this in lec06).

For now remember $\Phi(N) = \Phi(pq) = (p - 1)(q - 1)$.

It holds $\mathbb{Z}_N = \{0, 1, 2, \ldots, N - 1\}$ where $N$ is a positive integer.
Quiz Question!

Go to: http://socrative.com/
and use the code below or scan the QR code.
Student Login
Classroom: CRYPTOCHALMERS

RSA Encryption

Suppose that $N = 77$ and $e = 3$.
Can you encrypt the message $m = 75$?

Hint: $75 \equiv -2 \pmod{77}$

It holds:

$c = m^e \pmod{N}$

$= 75^3 \pmod{77}$

$\equiv (-2)^3 \pmod{77} \equiv -8 \pmod{77}$

$\equiv 77 - 8 \pmod{77} \equiv 69 \pmod{77}$
Security of the RSA Public Key

Hard to Compute

pk

Easy to Compute

sk

Where does the computational hardness come from?

- It is easy to compute $e^{-1}$ if one knows $p, q$ (or even $\Phi(N)$)...
  (More info about this in a few slides.)
- ... but not if one only knows $N = pq$
Reminder: What is a ‘good’ cipher?

SEEN in LEC02

Recall - Principles of Modern Cryptography

Describe clearly what you want to achieve so that you know if and when you have achieved it!

- Formal Definitions
- Precise Assumptions
- Proof of Security
Computationally hard problems - The Factoring Problem

- Modern cryptography is often based on the **assumption** that a certain problem cannot be solved efficiently *i.e.*, not in polynomial time (**computationally hard problem**).

**The Factoring Problem**

- Given a composite integer $N$, the factoring problem is finding its decomposition to prime factors.

  **Example:** in RSA the factoring problem is to find $p$ and $q$ such that $pq = N$ given only $N$.

---

**Best known algorithm (NFS) Number Field Sieve**

Run time $\exp(\tilde{O}(\sqrt[3]{n}))$ for $n$-bit integer.

**Current world record:** RSA-768 bits (232 digits)

- **Work:** two years on hundreds of machines.
- **Factoring** a 1024-bit integer: about 1000 times harder
  \[ \Rightarrow \text{likely possible this decade} \Rightarrow \text{Better to use 2048 bits integers} \]
How hard is the factoring problem? A closer look in complexity.
Reminder from the complexity/algorithms course.

- **PSPACE**: The problems that require polynomial space to be solved.
- **P problems**: everything that can be solved in polynomial time.
  - 'in polynomial time?': there exists a polynomial \( p() \) such that the algorithm runs in time \( p(n) \), where \( n \) is the length of the algorithm's input.
- **NP**: The class of all problems for which we do not know if there is a polynomial time algorithm to solve them.
- **NP complete**: Every other problem in NP can be reduced to this problem in polynomial time.
- **BQP**: The class of all problems that admit an efficient solution by quantum computers. BQP stands for bounded-error quantum polynomial time

⇒ *What about the factoring problem? It is an NP and BQP problem!*
Has the Factoring Problem been solved with quantum computing?

Shor’s algorithm

- A quantum algorithm for integer factorisation proposed in 1995 by Peter Shor.
- It solves the problem: Given an integer $N$, find its prime factors.
- Runs in polynomial time on quantum computers.
- In 2001, the algorithm factored 15 into $3 \times 5$.
- In 2012, the algorithm factored 21 into $3 \times 7$.

Factoring larger numbers will still take some time ...
Size of RSA prime numbers

How many prime numbers are there?
There are infinitely many prime numbers! (Theorem proven by Euclid 300 B.C.)

What is the largest prime number we know?
The largest known prime number (since 2016) is $2^{74207281} - 1$, which has 22,338,618 decimal digits. This means that if we typed it in a 12-point type, the number would be more than 80km long!!

How fast can we factor them?
In 2009 a collaboration of several research institutions factorised a 768-bit number with 232 decimal digits. The computing time is 2000 years on a single-core 2.2 GHz AMD Opteron.
Useful Concepts from Number Theory for PKC

PKC stands for Public Key Cryptography

Lecture 4
Modular arithmetics

Lecture 5
Extended Euclidean Algorithm
Prime Numbers
Primality Testing

Lecture 6
Group Theory
Fermat and Euler Theorems
Euler Phi (Totient) Function
Chinese Remainder Theorem
Discrete Logarithms

with Wissam Aoudi
Reminder: Modular Arithmetics

How to compute the “a \(\text{mod} \ N\)”?

1. Find the largest integer \(q\) s.t. \(a > qN\).
2. Compute \(r = a - qN\)
3. Then, \(r \equiv a \ (\text{mod} \ N)\)

Properties for Modular Arithmetics

For Integers in \(\mathbb{Z}_N\) where \(\mathbb{Z}_N = \{0, 1, 2, \ldots n - 1\}\)

- **Commutative:**
  \[w + x \ (\text{mod} \ N) = (x + w) \ (\text{mod} \ N)\]
  \[w \cdot x \ (\text{mod} \ N) = (x \cdot w) \ (\text{mod} \ N)\]

- **Associative:**
  \[[(w + x) + y] \ (\text{mod} \ N) = [w + (x + y)] \ (\text{mod} \ n)\]
  \[[(w \cdot x) \cdot y] \ (\text{mod} \ N) = [w \cdot (x \cdot y)] \ (\text{mod} \ n)\]

- **Distributive:**
  \[[(w \cdot (x + y)) \ (\text{mod} \ N) = [(w \cdot x) + (w \cdot y)] \ (\text{mod} \ n)\]

- **Identities:**
  \[(0 + w) \ \text{mod} \ n = w \ (\text{mod} \ N)\]
  \[(1 \cdot w) \ \text{mod} \ n = w \ (\text{mod} \ N)\]
How to Compute a Modular Inverse?

- In RSA the private key \(d\) is the inverse of the public key \(e\) i.e., \(d = e^{-1} \pmod{\Phi(N)}\).

**Question:** Does a number always have an inverse?

**Lemma:** \(a\) has an inverse in \(\mathbb{Z}_N\) if and only if \(\text{GCD}(a, N) = 1\)

**Question:** But how can I compute the GCD?
- Using the Euclidean Algorithm!

**GCD** stands for Greatest Common Divisor i.e., the largest positive integer that divides \(a\) and \(N\).

---

**Euclidean Algorithm:**

**Input:** Integers \(a, b\) with \(a \geq b \geq 0\).

**Output:** Integer \(d\) that is the greatest common divisor of \(a\) and \(b\)

- **Step 1:** Find \(q_1, r_1 \in \{0, 1, 2, \ldots\}\) such that:
  \[a = q_1b + r_1\] and \(r_1 < b\)

- **Step 2:** Find \(q_2, r_2 \in \{0, 1, 2, \ldots\}\) such that:
  \[b = q_2r_1 + r_2\] and \(r_2 < r_1\)

- **Step 3:** Do: Find \(q_i, r_i \in \{0, 1, 2, \ldots\}\) such that:
  \[r_{i-2} = q_ir_{i-1} + r_i\] and \(r_i < r_{i-1}\)

  until \(r_i = 0\)

- **Step 4:** \(d = r_{i-1}\) (the second-last remainder) is the GCD between \(a\) and \(b\)
How to Compute a Modular Inverse? - The Extended Euclidean Algorithm

Useful for the exercises/exams!

**Euclidean Algorithm:**

**Input:** Integers \(a, b\) with \(a \geq b \geq 0\).

**Output:** Integer \(d\) that is the **greatest common divisor** of \(a\) and \(b\)

- **Step 1:** Find \(q_1, r_1 \in \{0, 1, 2 \ldots\}\) such that:
  \[a = q_1 b + r_1\] and \(r_1 < b\)

- **Step 2:** Find \(q_2, r_2 \in \{0, 1, 2, \ldots\}\) such that:
  \[b = q_2 r_1 + r_2\] and \(r_2 < r_1\)

- **Step 3:** Do: Find \(q_i, r_i \in \{0, 1, 2, \ldots\}\) such that:
  \[r_{i-2} = q_i r_{i-1} + r_i\] and \(r_i < r_{i-1}\)
  until \(r_i = 0\)

- **Step 4:** \(d = r_{i-1}\) (the second-last remainder) is the **GCD** between \(a\) and \(b\)

**Example**

\[a = q_1 b + r_1\]

\[35 = q_1 6 + r_1\]

\[35 = (5) 6 + 5\]

\[6 = (1) 5 + 1\]

\[5 = (5) 1 + 0\] the **GCD** of 35 and 6

**Extended Euclidean Algorithm**

Write the GCD as a combination of \(a\) and \(b\)

\[\rightarrow 6 = (1) 5 + 1\]

\[6 - (1) 5 = 1\]

\[6 - (1)(35 - 5 \times 6) = 1\]

\[-(1) 35 + (5 + 1) 6 = 1\]

\[-1 \equiv 5 \pmod{6}\] is the **inverse** of 35 \((\text{mod } 6)\)

\[6 \pmod{35}\] is the **inverse** of 6 \((\text{mod } 35)\)
How to compute a modular inverse? Extended Euclidean Algorithm

Useful for the exercises/exams!

**Inverse mod \( p \)**

If \( x \) is an integer and \( p \) a prime then the inverse of \( x \) \((\text{mod } p)\) is defined as the number \( y \) \((\text{mod } p)\) such that it holds \( x \cdot y = 1 \pmod{p} \).

Usually the inverse of \( x \) is denoted as \( y = x^{-1} \pmod{p} \).

How to find the inverse of \( x \) \((\text{mod } p)\)?

Use the Extended Euclidean Algorithm!

**Extended Euclidean Algorithm:**

**Input:** Integers \( x \) and prime number \( p \).

**Output:** Integer \( y \) \((\text{mod } p)\) such that \( x \cdot y \equiv 1 \pmod{p} \).

- **Step 1:** Compute the Euclidean Algorithm between \( x \) and \( p \).
- **Step 2:** Find an equation of the form \( 1 = r_{i−2} − q_i r_{i−1} \).
- **Step 3:** Read the equation ‘reversely’ and write each remainder as a combination of the previous remainders until you reach an equation of the form \( 1 = x \cdot y + p \cdot m \)
- **Step 4:** From the previous equation we get \( x \cdot y = 1 \pmod{p} \), thus \( y \) is the modular inverse of \( x \) modulus \( p \).

**Attention:** the EEA works also when \( p \) is not prime! It can be used to find the inverse of any \( x \) \((\text{mod } n)\) (for general \( n \)) everytime \( x \) and \( n \) are co-prime, i.e., \( \text{GCD}(x, n) = 1 \).
How can I check if a number is prime? Primality Tests

Primality Test

- **Primality test**: an algorithm for determining whether an input number is a prime number or not.
- **Running time**: Polynomial in the size of the input.
- **Output**: prime/non-prime

**Remember**: All the algorithms for *primality testing* we will see are *probabilistic*!

What does this mean?

- If the algorithm outputs *prime*, this means that the input number has a **high probability** of being a **prime number**.
- If the algorithm outputs *non-prime* then the input number is **for sure** composite (**non-prime**).
Fermat Primality Test

Useful for the exercises/exams!

- **Inputs:** a positive integer \( n \) (\( n > 2 \)) to test for primality
  a positive integer \( k \) corresponding to number of iterations used in the primality test:
- **Output:** non-prime if \( n \) is composite, otherwise probably prime!

**Fermat Little Theorem**
If \( n \) is a prime number and \( a \neq 0 \), then it holds: \( a^{n-1} \equiv 1 \pmod{n} \)

Check an interesting video on Fermat’s primality test! [https://goo.gl/xuvxF](https://goo.gl/xuvxF)
A flaw of Fermat’s primality test: Carmichael numbers!

- There are (rare) non-prime numbers \( n \), for which it holds:
  \[ a^{n-1} \equiv 1 \pmod{n} \]
  for all \( a \in \{1, 2, \ldots, n - 1\} \)
  s.t. \( \gcd(a, n) = 1 \)

- They are called Carmichael numbers!
- The three smallest Carmichael numbers are 561 (\(3 \times 11 \times 17\)), 1105, 1729
- Proven to exist infinitely many Carmichael numbers!
- Although the Carmicheal numbers are rare, they are many enough to make Fermat’s primality test unsuitable for cryptography!
- Instead the Miller-Rabin algorithm is used (see next slide)!
Miller Rabin primality test

- **Inputs**: a positive integer $n$ ($n > 3$) to test for primality
  a positive integer $k$ corresponding to number of iterations used in the primality test:
- **Output**: non-prime if $n$ is composite, otherwise probably prime!

At most half of $a$s are misleading, i.e., the conditions are satisfied while is $n$ non-prime. So if I repeat the test $k$ times then the probability that $n$ is non-prime is $2^{-k}$. 

November 14, 2017 23 / 32
Textbook RSA: security

Is textbook RSA secure?

- To recover \( m \) the attacker has to compute \( m \) from \( m^e = c \pmod{N} \)
  \[ \Downarrow \]
  This is equivalent to solving the Dlog problem! More info on this lec07.
- Or to compute the decryption secret key \( d \) from the public key \( e \)
  \[ \Downarrow \]
  This is equivalent to finding a factorisation of \( N = pq \)

Can an attacker recover \( m \) in some other way?

YES!

Textbook RSA is not semantically secure and many attacks exist against it!!
Textbook RSA: security

Textbook RSA is deterministic!

*i.e.*, for a **fixed** key pair \((pk, sk)\), a message \(m\), always encrypts to the **same ciphertext**.

**Vulnerable to many attacks!**

**Brute Force attacks:** Since textbook RSA is **deterministic**, if the message \(m\) is chosen from a small list of possible values, then it is possible to determine \(m\) from the ciphertext \(c = m^e \pmod{N}\) by trying each value of \(m\).

**Question:** What if the message space is large, is it secure? **NO!**

**Other attacks:**

- **Meet-in-the-middle** attack (next slide).
- **Chosen Plaintext** attacks (we can distinguish \(m\) from \(m'\)) (More info in **lec07**).
- **Chosen Ciphertext** attacks.
Textbook RSA: a Meet-in-the-Middle attack

Web Browser

Client Hello

Server hello \((e, N)\)

\[ c = E_{RSA}(k) = k^e \pmod{N} \]

Web Server

Random session-key \(k\)

Let us suppose that \(k\) has 64 bits and that:

\[ k = k_1 \cdot k_2, \text{ where } k_1, k_2 < 2^{34} \] (this happens 20% of the times)

The attacker sees:

\[ k^e = c \Rightarrow (k_1 k_2)^e = c, \text{ and } N, e \]

**Quiz Question:** What should the attacker do to recover the key?

A **meet-in-the-middle** attack! \((k_1)^e = \frac{c}{k_2^e}\)

\[
\begin{array}{c|c|c}
(k_0)^e & \frac{c}{(k_0)^e} \\
(k_1)^e & \frac{c}{(k_1)^e} \\
(k_i)^e & \frac{c}{(k_j)^e} \\
& \vdots \\
(k_{2^{34}})^e & \frac{c}{(k_{2^{34}})^e}
\end{array}
\]

**Step 1:** The attacker can compute the table:

**Step 2:** ... and look for a match: \((k_i)^e = \frac{c}{(k_j)^e}\)

If he finds a match then he knows that \(k_i = k_2\) and \(k_j = k_1\) (after about \(2^{40}\) computations a match is found).
Improving Textbook RSA

Security depends on this step

- PKCS1 mode 2 (Public Key Cryptography Standard v1): some random padding in performed the original plaintext Widely deployed (e.g., HTTPS) **badly broken** (1998 Bleichenbacher attack)!

- RSA OAEP (Optimal Asymmetric Encryption Padding) proposed by Bellare & Rogaway in 1994 (RFC2437). Important but tedious to present in the lecture. Employs random padding and hash functions.
Quiz Questions! Requirements for Public Key Cryptography

Go to: http://socrative.com/
and use the code below or scan the QR code.
Student Login
Classroom: CRYPTOCHALMERS

1. It is computationally easy for a party A (Alice) to generate a pair of keys (pk_A, sk_A).
2. It is computationally easy for a sender B Bob to encrypt a message m using pk_A.
3. It is computationally easy for a receiver A (Alice) to decrypt a ciphertext c using sk_A.
4. It is computationally hard for an attacker to recover sk_A knowing pk_A.
5. It is computationally hard for an adversary who knows pk_A and a ciphertext c to recover the plaintext message m corresponding to c (i.e., decrypt c).
Things to Remember

- How to encrypt decrypt with textbook RSA?
- The factoring problem!
- Finding an inverse with the Extended Euclidean Algorithm.
- Fermat’s primality test.
- Miller- Rabin primality test
- Is textbook RSA secure?
- Describe the Meet-in-the-Middle attack against textbook RSA!
Home Assignment 4 (Programming Assignment)

The programming assignment is online. You are encouraged to work in pairs!

A Math Library for Cryptography (10 points)
In this assignment you will implement a library that provides a number of mathematical functions commonly used in cryptography, such as Euler's Phi Function (Totient), the Extended Euclidean Algorithm and some Primality Test. Your library needs to successfully pass a test suite that we provide.

Special Soundness of Fiat-Shamir sigma-protocol (10 points)
We eavesdropped on a number of Fiat-Shamir protocol runs and we found that the same nonce was used twice! Due to the special soundness property you should now be able to retrieve the secret key used in the protocol.

Decryption CBC with simple XOR (10 points)
We intercepted a message that was encrypted using cipher-block chaining. We also know the plain-text value of the first block. Can you reconstruct the complete plain-text message?

Attacking RSA (20 points)
This attack applies to the case in which the same message is encrypted using RSA to three different recipients. The enablers of the attack are (1) all recipients have the same public key \((e = 3)\) and (2) the recipients have different modulus \((N_1, N_2, N_3)\) that are coprime. Your goal in this assignment is to use the three eavesdropped ciphertexts and recover the secret message! The message you will recover is an ASCII encoding of a name you should know :)

Attacking ElGamal (20 points)
Thanks to its random component, two ElGamal encryptions of the same message can look completely different. However, this also makes the strength of the encryption depend on the random number generation. In this assignment you will attack ElGamal under a weak number generator.

The deadline is 15/12/2017!
Attention Changes in the Schedule!

<table>
<thead>
<tr>
<th>Tuesday 21/11/2017</th>
<th>Friday 17/11/2017</th>
</tr>
</thead>
<tbody>
<tr>
<td>08:00-10:00 Lecture 6</td>
<td><strong>Cancelled</strong> due to EU Mötet</td>
</tr>
<tr>
<td>10:00-12:00 Lecture 7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Friday 24/11/2017</th>
</tr>
</thead>
<tbody>
<tr>
<td>08:00-10:00 Exercise Session</td>
</tr>
<tr>
<td>10:00-12:00 Lecture 8</td>
</tr>
</tbody>
</table>
References:

- “Introduction to Modern Cryptography”, Lindell and Katz (Chapter 4.1.1, 4.1.2, 8.2.2, 11.2, 11.5.1, B.1-B.2)

Thank you for your attention!